Vysoká škola báňská – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Návrh kontejnerové virtualizace v linuxu

Design of Container Virtualization on Linux

2018 **Bc. Vít Zakopal**

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra telekomunikační techniky

Zadání diplomové práce

Student: **Bc. Vít Zakopal**

Studijní program: N2647 Informační a komunikační technologie

Studijní obor: 2601T013 Telekomunikační technika

Téma: Návrh kontejnerové virtualizace v linuxu
Design of Container Virtualization on Linux

Jazyk vypracování: čeština

Zásady pro vypracování:

Virtualizace umožňuje lepší využívání zdrojů a je také vhodná pro testování nového softwaru. Cílem diplomové práce je navrhnout řešení, postavené na kontejnerové virtualizaci v linuxu a následně provést výkonnostní testování. Jako virtualizační nástroje může být využito LXC, LXD, docker apod.

Řešení práce musí splňovat následující body:
1. Studium a popis virtualizačních technik.
2. Porovnání jednotlivých nástrojů pro kontejnerovou virtualizaci.
3. Instalace a konfigurace vybraných nástrojů.
4. Ověření a testování virtuálních serverů.
5. Výkonnostní testy navrženého řešení.

Seznam doporučené odborné literatury:

[1]PORTNOY, Matthew. *Virtualization Essentials.* Sybex; 1 edition 2012. ISBN: 1118176715
[2]MATTHIAS, Karl. *Docker: Up & Running.* O'Reilly Media; 1 edition 2015. ISBN-13: 978-1491917572

Formální náležitosti a rozsah diplomové práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí diplomové práce: **Ing. Pavel Nevlud**

Datum zadání: 01.09.2016
Datum odevzdání: 30.04.2018

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA Fakulta elektrotechniky a informatiky 1D16 OSTRAVA

doc. Ing. Miroslav Vozňák, Ph.D.
vedoucí katedry

prof. Ing. Pavel Brandštetter, CSc.
děkan fakulty

Prohlášení studenta

Prohlašuji, že jsem tuto bakalářskou/diplomovou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě dne: *24. dubna 2018*

podpis studenta

Poděkování

Rád bych na tomto místě poděkoval Ing. Pavlovi Nevludovi za vedení mé diplomové práce a především za velkou odbornou pomoc a konzultaci při jejím vytváření.

Abstrakt

Cílem této diplomové práce je návrh a realizace řešení postavené na kontejnerové virtualizaci v Linuxu. Práce popisuje problematiku virtualizačních technik a názorné rozdíly mezi nimi. V práci jsou popsány aktuálně nejpopulárnější a nejpoužívanější nástroje pro kontejnerovou virtualizaci v Linuxu, konkrétně Docker, LXC a LXD včetně jejich porovnání. Je zde podrobně popsána jejich instalace a následná konfigurace vzhledem k navrženému řešení, které počítá s vytvořením dvou webových serverů, kdy každý z nich má svůj vlastní kontejner a dostatek systémových prostředků pro funkčnost. Nadále jsou zde ukázky funkčnosti kontejnerů a virtualních serverů v nich obsažených. V závěru práce jsou virtuální servery každého z nástrojů podrobeny testování benchmark testem programu wkr a výstupem je grafické zobrazení čerpání jednotlivých zdrojů pomocí monitorovacího nástroje Monitorix.

Klíčová slova

Docker; kontejnery; Linux; LXC; LXD; Monitorix; Nginx; Virtualizace; webový server

Abstract

The main goal of this diploma thesis is to design and implement a solution based on container virtualization in Linux. The thesis describes the problematics of virtualization techniques and the visual differences between them. This thesis describes currently the most popular and most widely used container virtualization tools in Linux, namely Docker, LXC and LXD, including their comparison. It describes their installation and subsequent configuration in relation to the proposed solution, which involves creating two web servers, each of which has its own container and enough system resources for functionality. There are examples of the functionality of the containers and virtual servers contained therein. At the end of the work, the virtual servers of each tool are subjected to testing by the wkr benchmark test, and the output is a graphical view of individual resource draws using the Monitorix monitoring tool.

Keywords

Docker; containers; Linux; LXC; LXD; Monitorix; Nginx; Virtualization; web server

Seznam použitých zkratek

Zkratka	Význam
API	Application Programming Interface
CGI	Common Gateway Interface
Cgroups	Control Groups
CLI	Command Line Interface
CPU	Central Processing Unit
CRIU	Checkpoint/Restore In Userspace
DHCP	Dynamic Host Configuration Protocol
FTP	File Transfer Protocol
HTTP	Hypertext Transfer Protocol
I/O	Input/Output
IMAP	Internet Message Access Protocol
IP	Internet Protocol
IPC	Inter-Process Communication
IT	Information Technology
Libs	Libraries
LTS	Long Term Support
LXC	Linux Containers
MAC	Media Access Control
MB	Megabyte
MM	Memory Management
NAT	Network Address Translation
OEM	Original Equipment Manufacturer
OpenVZ	Open VirtualiZation
OS	Operating System
PaaS	Platform as a Service
PID	Process Identifier
POP	Post Office Protocol

SMTP	Simple Mail Transfer Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VM	Virtual Machine
VMM	Virtual Machnine Manager
WWW	World Wide Web
ZFS	Zettabyte File System

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Virtualizace je dnes v informatice velký pojem a řadí se mezi jednu z nejpoužívanějších metod pro práci s informačními technologiemi. Můžeme už virtualizatovat téměř cokoliv - servery, úložiště, sítě, desktopy nebo i aplikace. Umožňuje nám to lepší využívání zdrojů a je také vhodná pro testování nového softwaru. Kontejnerová virtualizace je něco nového, co se začalo rozvíjet až v posledních letech a mezi její hlavní vlastnosti patří menší náročnost na systém a efektivnější využívání fyzických zdrojů. Cílem této práce je zaměření na kontejnerovou virtualizaci a její využití s následným testováním.

První kapitola je věnována přehledu virtualizačních technik. Je zde celkový přehled a konkrétní popis virtualizačních technik, jmenovitě emulace, virtualizace na úrovni operačního systému, paravirtualizace a plná virtualizace. Součástí jsou také názorné přehledy architektury pro snazší rozeznávání rozdílů.

Druhá kapitola je věnována vybraným nástrojům pro kontejnerovou virtualizaci v Linuxu. Konkrétně se jedná o nástroje Docker, LXC a LXD. U každého z nástrojů je stručně popsána jejich případná historie, architektura, funkce a popis jejich celkové funkčnosti včetně vlastností vzhledem k možnostem vytvoření kontejnerů. Tyto popisy dohromady dávají obraz toho, jaké jsou mezi nimi rozdíly a vytvářejí tak jejich porovnání mezi sebou.

Třetí kapitola je věnována monitorovacím nástrojům. Vzhledem k tomu, že práce má obsahovat také výkonnostní testy řešení jednotlivých nástrojů, bylo zapotřebí nalézt vhodný nástroj pro grafické čerpání jednotlivých zdrojů, jako jsou např. procesor, paměť, síť atd. Jsou zde tedy popsány některé z nejpoužívanějších nástrojů pro tvorbu takovýchto grafů, jejich vlastnosti a možnosti nastavení, v neposlední řadě i ukázka vizualizace takového prostředí s grafy.

Čtvrtá kapitola je věnována návrhu řešení. Obsahuje konkrétně jaké softwarové a hardwarové vybavení včetně specifikací bylo použito. Nadále výběr služeb a to webové servery k umístění do kontejnerů zvlášť včetně tabulky s přidělenými odpovídajícími fyzickými zdroji. Součástí jsou také pro přehled způsoby síťování ve virtualizaci vzhledem k nástrojům pro kontejnerovou virtualizaci.

Pátá kapitola je věnována instalaci a konfiguraci vybraných nástrojů. V této kapitole se nachází podrobný popis instalace a konfigurace každého z nástrojů pro kontejnerovou virtualizaci v Linuxu, následováno instalací a konfigurací webových serverů do kontejnerů. Instalace jsou včetně příkazů a konfigurace jsou doplněny názornými ukázkami souborů. Toto je dále následováno ověřením a testováním funkčnosti jak kontejnerů, tak virtuálních serverů pomocí různých možností. Je zde i popis instalace, zprovoznění a návod k použití monitorovacího nástroje Monitorix pro vizualizaci grafů čerpání jednotlivých fyzických zdrojů.

Šestá kapitola je věnována výkonnostním testům. Kontejnery s virtuálními servery jsou zde podrobeny benchmark testu pomocí nástroje wrk po určitou dobu s definovaným množstvím otevřených http spojení. Po dobu testu je zapnut monitorovací nástroj, který po

uplynutí poskytuje grafy čerpání jednotlivých zdrojů v době spuštěného benchmarku. Tyto grafy jsou hlavním předmětem výkonnostních testů pro porovnání jednotlivých řešení s kontejnerovými nástroji.

Závěr je věnován zhodnocení dosažených výsledků a vyvození celkového výkonu nástrojů pro kontejnerovou virtualizaci v Linuxu.

1 Přehled virtualizačních technik

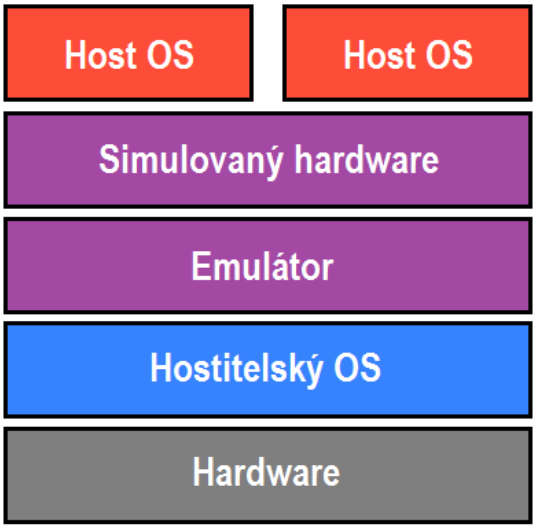
Při rozhodování o nejlepším přístupu k implementaci virtualizace je důležité mít jasné znalosti různých řešení virtualizace, která jsou v současnosti k dispozici. Účelem této kapitoly tedy je obecně popsat virtualizační techniky, které jsou dnes běžně používány, a to emulace, virtualizace na úrovni operačního systému, paravirtualizace a plná virtualiazace.

1.1 Emulace

Emulace je pravděpodobně virtualizační technika, na kterou většina myslí, když se přemýšlí o hardwarové virtualizaci. Jak vyplývá z jejího názvu, emulace je tam, kde hypervisor napodobuje určitý hardware, který představuje hostu VM, bez ohledu na to, co je skutečný fyzický hardware. Při použití emulace získáte klasický přínos „přenositelnosti VM“, což znamená, že libovolné zařízení VM může pracovat na libovolném hardwaru. To dává smysl, protože host VM vidí pouze emulovaný hardware, ne skutečný fyzický hypervisor.

Takže emulace má dobrou a širokou přenosnost, ale na úkor výkonu. Problémem je, že hypervisor musí vybrat „nejnižší společný jmenovatel“ při rozhodování o tom, jak se vypořádat s nejrůznějším fyzickým hardwarem. Takže toto je místo, kde získáte generickou grafickou kartu uvnitř vašeho VM, i když máte fyzicky třeba lepší a výkonnou grafickou kartu. A pokud host VM vidí pouze generickou kartu, pak je to vše, co může použít. Nebudete moci spustit výkonnou grafickou kartu nebo Aero nebo cokoli, pokud váš hypervisor jen emuluje něco obecného.

Emulace je také odpovědná za největší „výkonnostní trest“ při virtualizaci. Je to tak bráno, protože hypervisor potřebuje obdržet pokyny pro falešný hardware, který emuluje, a pak je přetransformuje na jakýkoli skutečný hardware.[1]



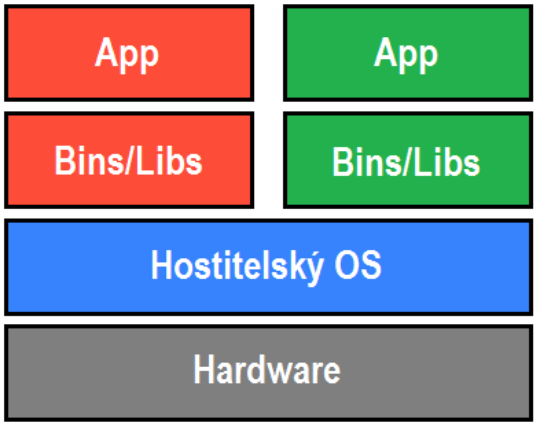
Obrázek 1.1: *Architektura emulace*

1.2 Virtualizace na úrovni operačního systému

Virtualizace na úrovni operačního systému, známá také jako kontejnerizace, odkazuje na funkci operačního systému, v níž jádro umožňuje existenci více izolovaných instancí uživatelského prostoru. Takové instance, nazvané jako kontejnery, oddíly, virtualizační enginy (VEs) nebo jaily (FreeBSD jail nebo chroot jail) mohou vypadat jako skutečné počítače z pohledu programů spuštěných v nich. Počítačový program běžící na obyčejném operačním systému vidí všechny zdroje (připojená zařízení, soubory a složky, síťové sdílení, výkon procesoru, vyčíslitelné hardwarové schopnosti) tohoto počítače. Programy běžící uvnitř kontejneru však mohou zobrazit pouze obsah kontejneru a zařízení, která jsou danému kontejneru přidělena.

V operačních systémech typu Unix lze tuto funkci vidět jako pokročilou implementaci standardního chrootového mechanismu, který změní zdánlivou kořenovou složku aktuálního spuštěného procesu a jeho součásti. Kromě mechanismů izolace, jádro často poskytuje funkce správy prostředků, které omezují dopad činností jednoho kontejneru na jiné kontejnery.

Virtualizace na úrovni operačního systému se běžně používá ve virtuálních hostitelských prostředích, kde je užitečná pro bezpečnou alokaci konečných hardwarových prostředků mezi velkým počtem vzájemně nedůvěřivých uživatelů. Správci systému jej mohou také použít pro konsolidaci hardwaru serveru přesunutím služeb na samostatných hostitelích do kontejnerů na jednom serveru. Jiné typické scénáře zahrnují oddělení několika programů do oddělených kontejnerů, které zlepšují zabezpečení, nezávislost hardwaru a přidané funkce správy zdrojů.[2]



Obrázek 1.2: *Architektura virtualizace na úrovni operačního systému*

1.2.1 Kontejnerová virtualizace

O kontejnerové virtualizaci se hovoří tehdy, když se v rámci jednoho operačního systému vytváří vzájemně oddělená prostředí, tzv. kontejnery. Lze tak na jednom stroji provozovat kupř. několik webových serverů, aniž by bylo nutné mít pro každý z nich nainstalovaný kompletní systém.[3]

Za výhody lze považovat:

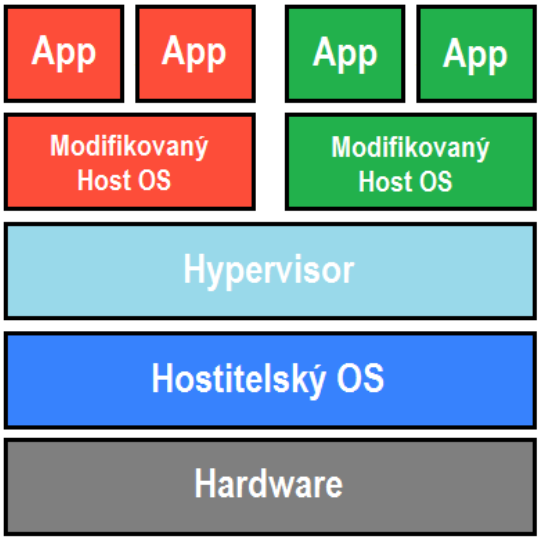
- Menší náročnost na systémové prostředky
- Menší technická náročnost - kontejnery se vytváří v rámci jedné instalace
- Nad hardware běží pouze jedno jádro - kontejnery využívají společnou MM, power management, scheduler atd.

Jako nevýhody můžeme brát:

- Nejde o opravdovou virtualizaci - používá se stejný kernel a oddělení není vlastně úplné
- Problém s technologiemi jako SELinux, AppArmor

1.3 Paravirtualizace

V paravirtualizaci je si host OS vědom rozhraní se základním hostitelským operačním systémem. Paravirtualizované jádro hostu chápe základní technologii hostitele a využívá tuto skutečnost. Vzhledem k tomu, že hostitelský operační systém nepředstírá hostovi sto procent, množství prostředků potřebných pro virtualizaci je výrazně sníženo. Kromě toho mohou paravirtualizované ovladače zařízení pro hosty propojit s hostitelským systémem a snížit režii. Myšlenkou paravirtualizace je snížit jak složitost, tak režii zapojenou do virtualizace. Díky paravirtualizaci jak host OS, tak i hostitelského operačního systému, jsou od hosta k hostitelskému operačnímu systému vynaloženy velmi náročné funkce. Host v podstatě volá speciální systémová volání, která pak umožňují tyto funkce spouštět v hostitelském operačním systému. Při použití systému, jako je Oracle VM, funguje hostitelský operační systém podobně jako host operační systém. Součástí jsou ovladače hardwarových zařízení rozhraní s vrstvou známé jako hypervisor. Hypervisor je také znám jako monitor virtuálního stroje (VMM). Existují dva typy hypervisorů. Hypervisor buď běží přímo na hostitelském hardwaru nebo hostovaný hypervisor běží v softwaru. [4]

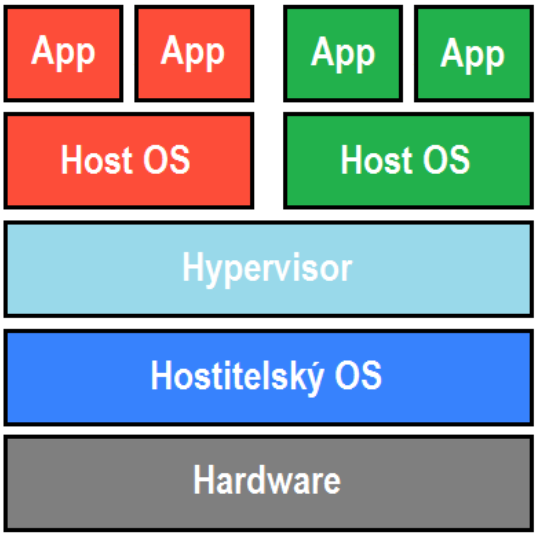


Obrázek 1.3: *Architektura paravirtualizace*

1.4 Plná virtualizace

Při plné virtualizaci softwaru je veškerý hardware simulován softwarovým programem. Každý ovladač zařízení a proces v host operačním systému „věří“, že běží na skutečném hardwaru, i když základní hardware je ve skutečnosti softwarový program.

Jednou z výhod plné virtualizace softwaru je, že na něm můžete spustit libovolný operační systém. Nezáleží na tom, zda daný operační systém chápe základní hardware hostitele nebo ne. Tak mohou v tomto prostředí fungovat starší operační systémy a speciální operační systémy. Architektura je velmi flexibilní, protože nepotřebujete zvláštní znalost OS nebo hardwaru.



Obrázek 1.4: *Architektura plné virtualizace*

Hardwarový subsystém OS zjišťuje hardware normálním způsobem. Věří, že hardware je opravdu hardware. Typy a funkce hardwaru, které objevují, jsou zpravidla poměrně obecné a nemusí být tak plnohodnotné jako skutečné hardwarové zařízení, i když je systém funkční. Další výhodou plné virtualizace softwaru je to, že nemusíte zakoupit žádný další hardware. Díky hardwarové virtualizaci softwaru musíte zakoupit hardware, který podporuje pokročilou technologii VM. Přestože tato technologie je součástí většiny systémů, které jsou dnes k dispozici, některé starší hardwarové nástroje tuto funkci nemají. Chcete-li tento starší hardware použít jako virtuální host, musíte použít buď plnou virtualizaci softwaru nebo paravirtualizaci.

Pouze hardwarově podporovaná virtualizace softwaru vyžaduje pokročilé hardwarové funkce VM, plná virtualizace softwaru ne. Kupříkladu Oracle VM a VMware ESX pracují se starším hardwarem, který nemá žádné speciální funkce procesoru. Tento typ virtualizace je výše zmíněná emulace. Bohužel plná virtualizace softwaru zvyšuje režii. Tato režie se přenáší na další pokyny a čas procesoru na hostu, což má za následek pomalejší systém a vyšší využití procesoru. Při plné virtualizaci softwaru jsou volání instrukcí procesoru zachycena nástrojem monitoru virtuálního stroje a potom emulována v softwarovém programu. Proto každá hardwarová instrukce, která by byla normálně řešena samotným hardwarem, je nyní

zpracovávána programem. Například, když ovladač diskového zařízení provede I / O volání na „virtuální disk“, software v systému VM jej zachytí, zpracuje a nakonec provede I / O na skutečný podkladový disk. Počet pokynů pro provádění I / O je značně vyšší. Při vytváření sítí vzniká ještě vyšší režie, protože v softwaru je simulován síťový přepínač. V závislosti na intenzitě síťové aktivity můžou být režijní náklady poměrně vysoké. Ve skutečnosti s vážně přetíženými hostitelskými systémy byste možná mohli vidět síťové zpoždění ze samotného virtuálního přepínače. Proto je velikost tak důležitá. [4]

2 Nástroje pro kontejnerovou virtualizaci

Virtualizačních nástrojů pro kontejnerovou virtualizaci existuje velké množství. Jelikož je ale virtualizace pomocí kontejnerů v Linuxu relativně nový trend, který se prosazuje až v několika posledních letech, pouze menší počet těchto nástrojů je používáno, vzhledem ke komunitám kolem nich. Mezi nejpopulárnější a nejpoužívanější se řadí Docker, následován LXC a LXD, které jsou také součástí zadání a pomocí nichž jsem realizoval svou praktickou část práce. Abychom porozuměli kontejnerové technologii, musíme začít s Linux Cgroups a Namespaces, oba jsou izolační koncepty v jádře Linuxu.

Namespaces: Koncept původně vyvinutý společností IBM, jmenný prostor systému Linux zabaluje sadu systémových prostředků a předkládá je procesům ve jmenném prostoru, což vypadá, jako by se věnovalo procesům.

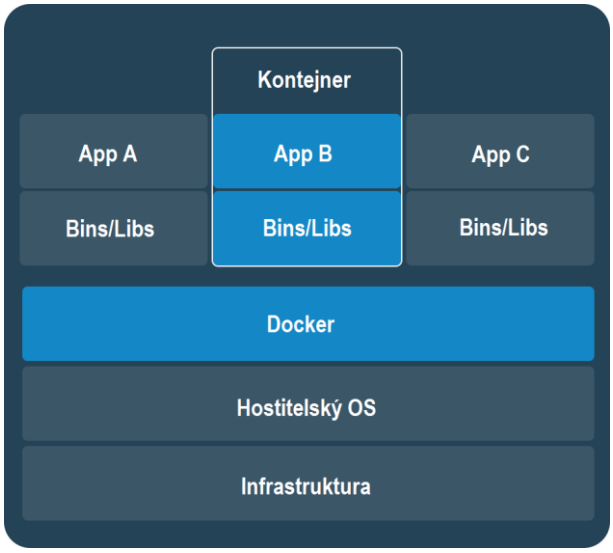
Cgroups: Původně jimi přispěla společnost Google, Cgroups je koncept Linuxového jádra, který řídí izolaci a využití systémových prostředků, jako CPU a paměť, pro skupinu procesů. Pokud máte například aplikaci, která využívá mnoho cyklů procesoru a paměti, například vědecké výpočetní aplikace, můžete aplikaci umístit do skupiny Cgroup, která omezuje využití procesoru a paměti.

Namespaces a Cgroups jsou nástroje pro správu prostředků v rámci systému Linux, které pomáhají izolovat systémové prostředky pro procesy. Společně tvoří základ moderních virtuálních kontejnerů.

2.1 Docker

Tento nástroj byl původně vytvářen jako interní projekt modelu platforma jako služba (PaaS) v rámci společnosti dotCloud. Docker představuje evoluci vlastní technologie dotCloud, která je sama postavena na dřívějších open-source projektech, jako jsou Cloudlets. Je primárně vyvinut pro Linux a nyní je stále vyvíjen jako open-source přímo společností Docker. Pomocí Dockeru můžete spravovat infrastrukturu stejným způsobem, jakým spravujete své aplikace. Tím, že využijete metodiky společnosti Docker pro rychlé doručení, testování a zavádění kódu, můžete významně snížit zpoždění mezi psaním kódu a jeho spuštěním ve výrobě.

Kontejner běží na Linuxu a sdílí jádro hostitelského počítače s dalšími kontejnery. Spouští se diskrétní proces, který nevyžaduje více paměti než kterýkoli jiný spustitelný soubor, takže je nenáročný.



Obrázek 2.1: *Architektura Docker*

Bins jsou soubory kódů čitelných počítačem v binárním formátu, které řídí procesor a procesor přímo s bity. Libs jsou funkce použitelné různými programy. [5]

2.1.1 Platforma

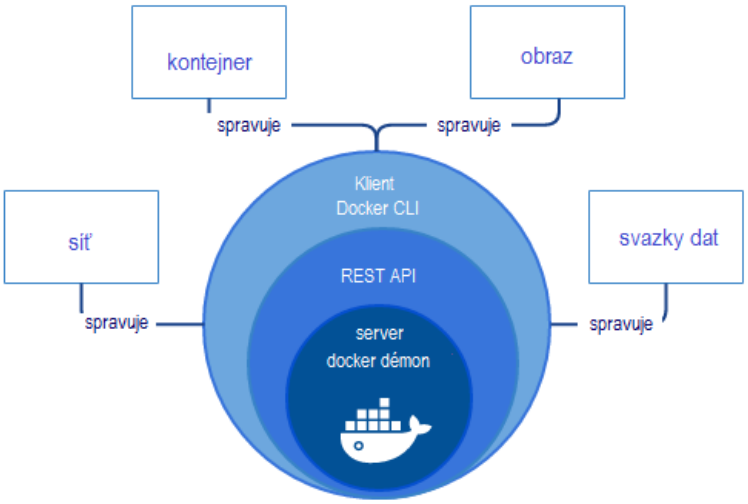
Poskytuje možnost zabalení a spuštění aplikace ve volně izolovaném prostředí nazývaném kontejner. Izolace a zabezpečení umožňují spustit více kontejnerů současně na daném počítači. Kontejnery jsou nenáročné, protože nepotřebují extra zatížení hypervisoru, ale běží přímo v jádře hostitelského počítače. To znamená, že můžeme spustit více kontejnerů v dané hardwarové kombinaci, než kdybychom používali virtuální počítače. Můžeme dokonce spustit Docker kontejnery v hostitelských strojích, které jsou ve skutečnosti virtuální stroje (VM).

Nadále poskytuje nástroj a platformu pro správu životního cyklu vašich kontejnerů. Lze tak vyvíjet svou aplikaci a její podpůrné komponenty pomocí kontejnerů. Kontejner se poté stává jednotkou pro distribuci a testování aplikace. A když je vše připraveno, nasadí se aplikace do produkčního prostředí jako kontejner nebo řízená služba. Funkčnost bude stejná, ať už produkční prostředí je lokální datové centrum, poskytovatel cloudu nebo kombinace obou těchto prostředí.

2.1.2 Engine

Docker engine je aplikace klient-server s těmito hlavními součástmi:

- Server, který je typem dlouhodobého programu nazvaného démon
- Aplikace REST API, která specifikuje rozhraní, které programy mohou používat ke komunikaci s démonem, a instruuje jej, co má dělat.
- Klient rozhraní příkazového řádku (CLI)



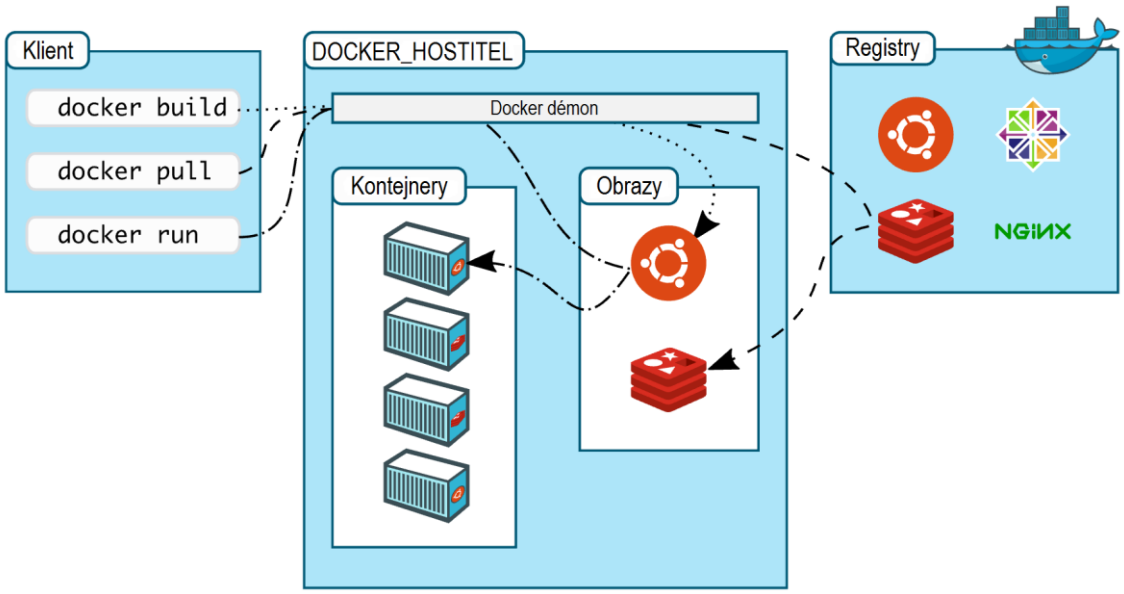
Obrázek 2.2: *Architektura Engine*

Docker CLI používá REST API pro ovládání nebo interakci s démonem prostřednictvím skriptování nebo přímých příkazů CLI. Mnoho dalších aplikací aplikace Docker používá základní rozhraní API a rozhraní CLI.

Démon vytváří a spravuje objekty Dockeru, jako obrazy, kontejnery, sítě a svazky.

2.1.3 **Architektura**

Docker používá architekturu klient-server. Klient Dockeru mluví s démonem Dockeru, který provádí vytváření, běh aplikace a distribuuje Docker kontejnery. Klient a démon mohou být spuštěny ve stejném systému nebo můžete připojit klienta k vzdálenému démonu Dockeru. Klient a démon Docker komunikují pomocí rozhraní REST API, přes UNIX sokety nebo síťové rozhraní.



Obrázek 2.3: *Architektura klient-server*

2.1.3.1 *Démon*

Docker démon (dockerd) naslouchá žádostem API a spravuje objekty Dockeru, jako jsou obrazy, kontejnery, sítě a svazky. Démon může také komunikovat s dalšími démony pro správu služeb Dockeru.

2.1.3.2 *Klient*

Klient Docker (docker) je primárním způsobem, jakým mnoho uživatelů Dockeru komunikuje s Dockerem. Při použití příkazů, jako je například „docker run“, klient odešle tyto příkazy do „dockerdu“, který je provede. Příkaz „docker“ používá rozhraní Docker API. Klient Docker může komunikovat s více než jedním démonem.

2.1.3.3 *Registry*

Docker registry ukládají obrazy Dockeru. Docker Hub a Docker Cloud jsou veřejné registry, které může použít každý, a Docker je nakonfigurován tak, aby ve výchozím nastavení nakonfiguroval obrazy ve službě Docker Hub. Dokonce můžete spustit vlastní soukromý registr. Pokud používáte Docker Datacenter (DDC), obsahuje poté Docker Trusted Registry (DTR).

Použijete-li příkazy „docker pull“ nebo „docker run“, požadované obrazy jsou vytaženy z konfigurovaného registru. Použijete-li příkaz „docker push“, váš obraz je přenesen do nakonfigurovaného registru.

Součástí je také možnost využít Docker Store, který vám umožní koupit či prodávat obrazy Dockeru nebo je bezplatně distribuovat. Například si můžete zakoupit obraz Dockeru obsahující aplikaci nebo službu od dodavatele softwaru a pomocí obrazu tak nasadit aplikaci do testovacích, stagingových a produkčních prostředí. Aplikaci můžete upgradovat vytažením nové verze obrazu a přesunem kontejnerů.

2.1.3.4 ***Obrazy***

Obraz (image) je šablona pouze pro čtení s pokyny pro vytvoření kontejneru Docker. Často je obraz založen na jiném obrazu s několika dalšími úpravami. Můžete například vytvořit obraz, který je založen na obrazu Ubuntu, ale nainstalujete webový server Apache a vaši aplikaci, stejně jako údaje o konfiguraci potřebné pro spuštění aplikace.

2.1.3.5 ***Kontejnery***

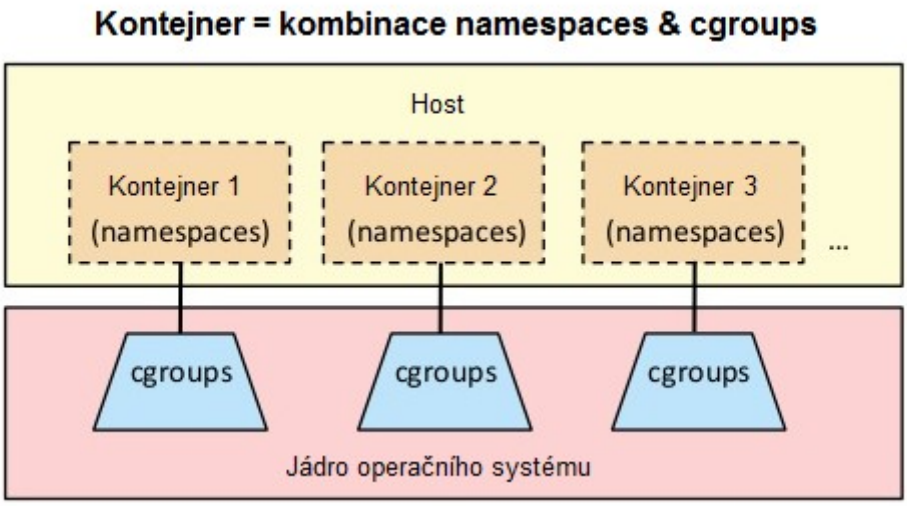
Kontejner je spouštěcí instancí obrazu. Je možné ho vytvořit, spustit, zastavit, přesunout nebo odstranit pomocí rozhraní Docker API nebo CLI. Můžete přidat kontejner do jedné nebo více sítí, připojit k němu úložiště nebo dokonce vytvořit nový obraz podle jeho aktuálního stavu.

Ve výchozím nastavení je kontejner relativně dobře izolován od ostatních kontejnerů a jeho hostitelského stroje. Můžete řídit, jak je izolovaná kontejnerová síť, úložiště nebo další podkladové subsystémy z jiných kontejnerů nebo z hostitelského počítače.

Kontejner je definován jeho obrazem stejně jako všechny možnosti konfigurace, které mu při vytváření nebo spuštění vytváříte. Po odstranění kontejneru zmizí všechny změny stavu, které nejsou uloženy v trvalém úložišti.

2.1.4 **Základní technologie**

Docker je napsán v programu Go a využívá několika funkcí jádra Linuxu, aby poskytl své funkce.



Obrázek 2.4: *Základ kontejneru*

2.1.4.1 ***Namespaces***

Docker používá technologii nazvanou namespaces, aby poskytla izolovaný pracovní prostor nazvaný kontejner. Při spuštění kontejneru vytvoří Docker sadu namespaces pro tento kontejner.

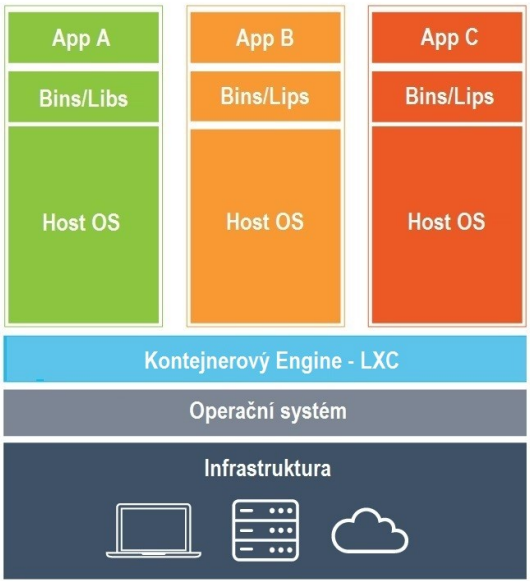
Tyto namespaces poskytují vrstvu izolace. Každý aspekt kontejneru běží v samostatném oboru namespaces a jeho přístup je omezen na tento daný jmenný prostor.

2.1.4.2 ***Control groups***

Docker Engine na Linuxu také spoléhá na jinou technologii nazývanou kontrolní skupiny (cgroups). Skupina cgroup omezuje aplikaci na konkrétní soubor prostředků. Řídicí skupiny umožňují Docker Engine sdílet dostupné zásoby hardwaru s kontejnery a volitelně vynutit omezení. Můžete například omezit dostupnou paměť pro konkrétní kontejner.

2.2 LXC

Linux Containers (LXC) je virtualizační metoda na úrovni operačního systému pro spouštění několika izolovaných systémů Linux (kontejnery) na jediném ovládacím hostiteli (hostitel LXC). Neposkytuje virtuální stroj, ale spíše poskytuje virtuální prostředí, které má vlastní procesor, paměť, blok I / O, síť atd. a mechanismus řízení zdrojů.



Obrázek 2.5: *Architektura LXC*

Mezi hlavní výhody LXC patří snadné ovládání virtuálního prostředí pomocí nástrojů uživatelského prostoru z hostitelského operačního systému, které vyžadují nižší režii než tradiční hypervisor a zvyšují přenositelnost jednotlivých aplikací a umožňují je distribuovat uvnitř kontejnerů.

Pokud si myslíte, že LXC zní hodně jako kontejnery Docker, je to proto, že LXC bývala základní technologií, která dělala Dockeru výchozí základ. Nedávno se však Docker vydal svým vlastním směrem a již není závislý na LXC. Přesto LXC byla původem revoluce kontejnerů před několika lety a principy LXC, ne-li kód LXC, zůstávají ústředním bodem pro vývoj kontejnerů. [6][7]

2.2.1 Funkce

LXC poskytuje virtualizaci na úrovni operačního systému prostřednictvím virtuálního prostředí, které má vlastní procesní a síťový prostor, namísto vytvoření úplného virtuálního stroje. LXC se spoléhá na funkci cgroups Linux jádra a spoléhá se také na další druhy funkcí pro izolaci jmenného prostoru, které byly vyvinuty a integrovány do hlavního jádra Linuxu.

Aktuální LXC používá následující funkce jádra, které obsahují procesy:

- Namespaces jádra (ipc, uts, mount, pid, network a user)
- Apparmor a SELinux profily
- Politiky Seccomp
- Chrooty (pomocí pivot_root)
- Funkce jádra
- CGroups (kontrolní skupiny)

Kontejnery LXC jsou často považovány za něco uprostřed mezi chrootem a plnohodnotným virtuálním strojem. Cílem LXC je vytvořit prostředí co nejblíže standardní instalaci Linuxu, ale bez nutnosti samostatného jádra.

2.2.2 Architektura

LXC je v současné době tvořen z několika samostatných komponent:

- Knihovna liblxc
- Několik jazykových vazeb pro rozhraní API jako python3, lua, Go, ruby, python2 a Haskell
- Sada standardních nástrojů pro ovládání kontejnerů
- Šablony k distribuci kontejnerů

2.2.3 Bezpečnost

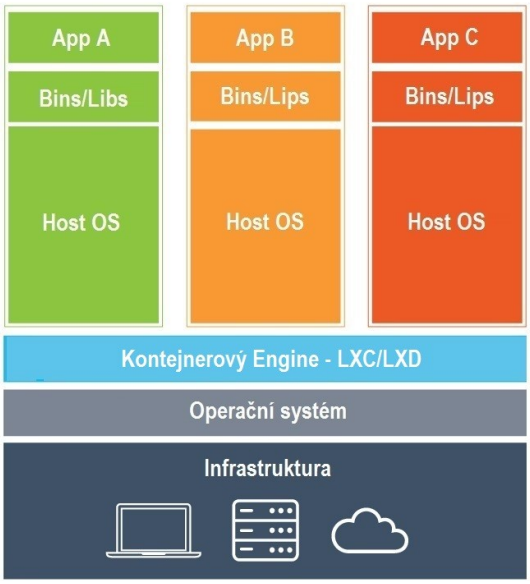
Původní kontejnery LXC nebyly tak bezpečné jako jiné virtualizační metody na úrovni operačního systému, jako je OpenVZ: v jádrech Linuxu ve starší verzi, kořenový uživatel hostujícího systému mohl spustit libovolný kód v hostitelském systému s oprávněním uživatele root, podobně jako chroot.

Počínaje nejnovější verzí LXC je možné provozovat kontejnery jako běžní uživatelé na hostiteli pomocí „neprivilegovaných kontejnerů“. Neprivilegované kontejnery jsou omezené tím, že nemají přímý přístup k hardwaru. Nicméně i privilegované kontejnery by měly zajistit odpovídající izolaci v bezpečnostním modelu LXC, pokud jsou správně nakonfigurovány.

2.3 LXD

LXD je open source rozšíření pro správu kontejneru pro linuxové kontejnery (LXC). LXD zlepšuje stávající funkce LXC a poskytuje nové funkce a funkcionalitu pro vytváření a správu kontejnerů Linux.

LXD je vlastně hypervisor nové generace pro Linux. LXD je navržen tak, aby „zvedl a posunul“ Linux virtuální stroje do kontejnerů bez úpravy aplikace nebo operací. Téměř všechny aplikace mohou být nainstalovány v kontejnerech LXD, aniž by bylo nutno zasahovat do aplikace, aby se spustila, protože kontejnery strojů LXD pracují stejně jako virtuální stroje. [8][9]



Obrázek 2.6: *Architektura LXD*

2.3.1 Vlastnosti

Některé z největších vlastností LXD jsou:

- Zabezpečení (neprivilegované kontejnery, omezení zdrojů)
- Škálovatelnost (z kontejnerů na notebooku až po tisíce výpočetních uzlů)
- Intuitivnost (jednoduché, jasné rozhraní API a ostré příkazové řádky)
- Založenost na obrazu (s širokou škálou distribucí Linux publikovaných denně)
- Podpora přenosu kontejnerů a obrazů mezi hostitelem (včetně živé migrace s CRIU)
- Pokročilá kontrola zdrojů (procesor, paměť, síťové I / O, blokové I / O, využití disku a zdroje jádra)
- Řízení sítě (tvorba a konfigurace mostů, tunely mezi hostiteli atd.)
- Správa úložišť (podpora více záloh pro ukládání dat, zásobníků a objemů úložišť)

2.3.2 Architektura

Jádro LXD je privilegovaný démon, který vystavuje REST API přes lokální socket Unixu i přes síť (pokud je zapnuto).

Klienti, jako je nástroj příkazového řádku dodávaný s LXD samotným, pak dělají vše prostřednictvím tohoto REST API. To znamená, že ať už komunikujete s místním hostitelem nebo vzdáleným serverem, funguje to stejně.

LXD pracuje na libovolné distribuci Linuxu. LXD upstream přímo udržuje balíčky Ubuntu a také vydává balíček, který lze použít s většinou populárních distribucí Linuxu.

2.3.3 Kontejnery

LXD kontejnery jsou stejné jako tradiční fyzické a virtuální stroje. Můžete se přihlásit vzdáleně, řídit hostující operační systém standardním způsobem, instalovat aplikace, zabezpečit je a obsluhovat je pomocí stejných nástrojů, ale ve výsledku získávat špatný výkon a hustotu kontejnerů. Hosté LXD se spustí během několika sekund a můžete spustit stovky na jednom serveru. Lze je připojit odděleně a bezpečně do sítí, kopírovat mezi hostitelemi a řídit vše přes čisté API rozhraní REST.

LXD používá LXC prostřednictvím liblxc a jeho Go pro vytvoření a správu kontejnerů. Je to v podstatě alternativa k nástrojům LXC a distribuční šabloně s přidanými funkcemi, které přicházejí z toho, že jsou ovládatelné v síti.

2.3.4 Integrace s OpenStack

Projekt „nova-lxd“ poskytuje OpenStack Nova plugin, který jednoduše integruje systémové kontejnery do běžného nasazení OpenStack.

Tímto způsobem uživatelé buď získají virtuální počítač nebo kontejner, jednoduše podle toho, jaký obraz nebo typ instance zvolí. Je zcela transparentní a pracuje s běžnými API OpenStack.

2.4 Srovnání nástrojů pro kontejnerovou virtualizaci

Na základě obecných informací se Docker jednoduše jeví jako nejlepší nástroj pro kontejnerovou virtualizaci. Oproti LXC/LXD má obrovské možnosti nastavení, což je výhoda především pro programátory. Ale za největší přednost považuji již obsáhlou databázi různých obrazů, ať už komunitních nebo komerčních. Také jednoduchá možnost zakoupení již hotového kontejneru s nějakou službou a jeho následné zavedení do provozu je dle mého názoru budoucnost v distribuci těchto služeb, a celkové zjednodušení zaběhlých procesů vytváření.

3 Monitorovací nástroje

Vzhledem k tomu, že součástí zadání jsou také výkonnostní testy navrženého řešení, bylo potřeba nalézt monitorovací nástroj, pomocí něhož je možnost toto realizovat. Ideální je použít takový nástroj, který poskytne grafické zobrazení čerpání jednotlivých zdrojů (cpu, paměť, síť atd.). Takovýchto nástrojů je celá řada a tak si ukážeme alespoň některé, načež jeden z nich bude posléze použit pro již zmíněné výkonnostní testy.

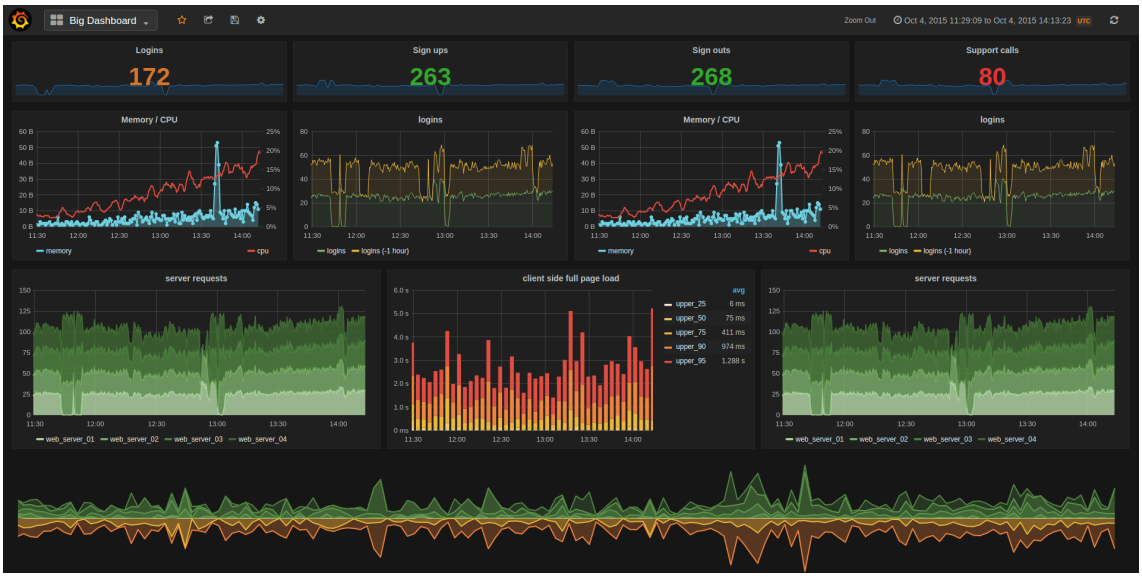
3.1 Grafana

Grafana je open-source, univerzální deska přístrojů s obecným účelem a vytvářeč grafů, který běží jako webová aplikace. Podporuje grafit, InfluxDB nebo opentsdb jako backends.

Vizualizuje rychlé a flexibilní grafy na straně klienta s mnoha možnostmi. Poskytuje panelové pluginy pro mnoho různých způsobů zobrazení metrik a protokolů.

Lze vizuálně definovat pravidla výstrah pro nejdůležitější metriky. Když se stav upozornění změní, odešle oznámení. Obdržíte například emailové upozornění se specifikací, který limit byl překročen.

Filtry Ad-hoc vám umožní vytvářet nové filtry klíčů/hodnot za běhu, které se automaticky použijí na všechny dotazy, které tento zdroj dat používají. [10]

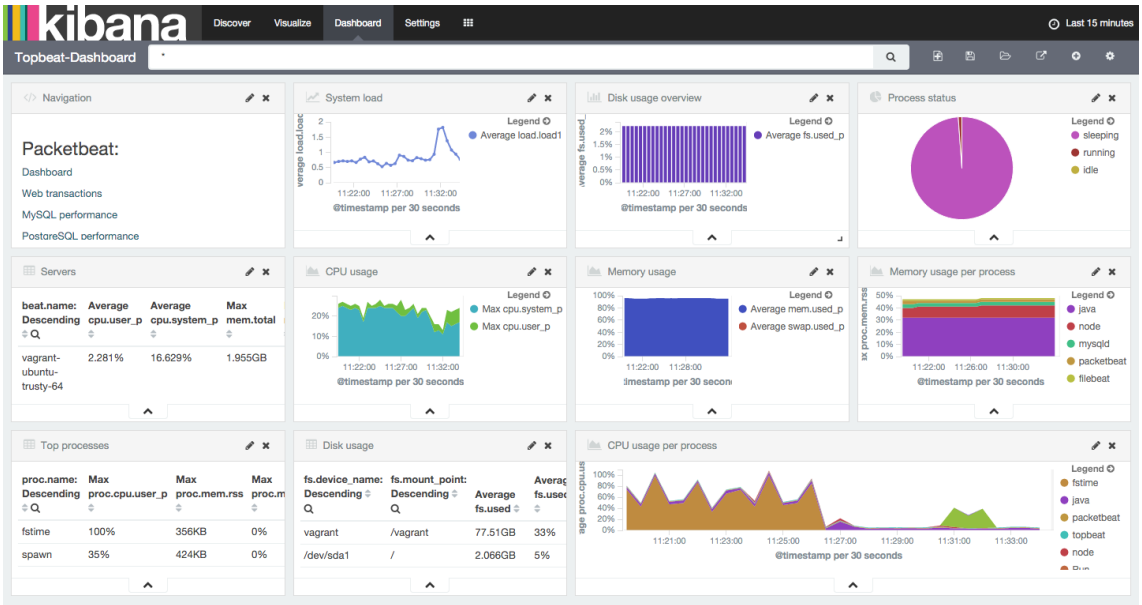


Obrázek 3.1: *Ukázka Grafana*

3.2 Kibana

Jedná se o open-source plugin pro vizualizaci dat pro Elasticsearch. Poskytuje vizualizační schopnosti nad obsah indexovaný na clusteru Elasticsearchu. Uživatelé mohou vytvářet grafy, řádky a rozptýlené grafy nebo grafy a mapy na velkém objemu dat.

Kombinace Elasticsearch, Logstash a Kibana, označovaná jako „Elastic Stack“ (dříve „ELK stack“), je k dispozici jako produkt nebo služba. Logstash poskytuje vstupní proud dat pro Elastic na ukládání a vyhledávání a Kibana přistupuje k datům pro vizualizace, jako jsou například dashboardy. [11]



Obrázek 3.2: *Ukázka Kibana*

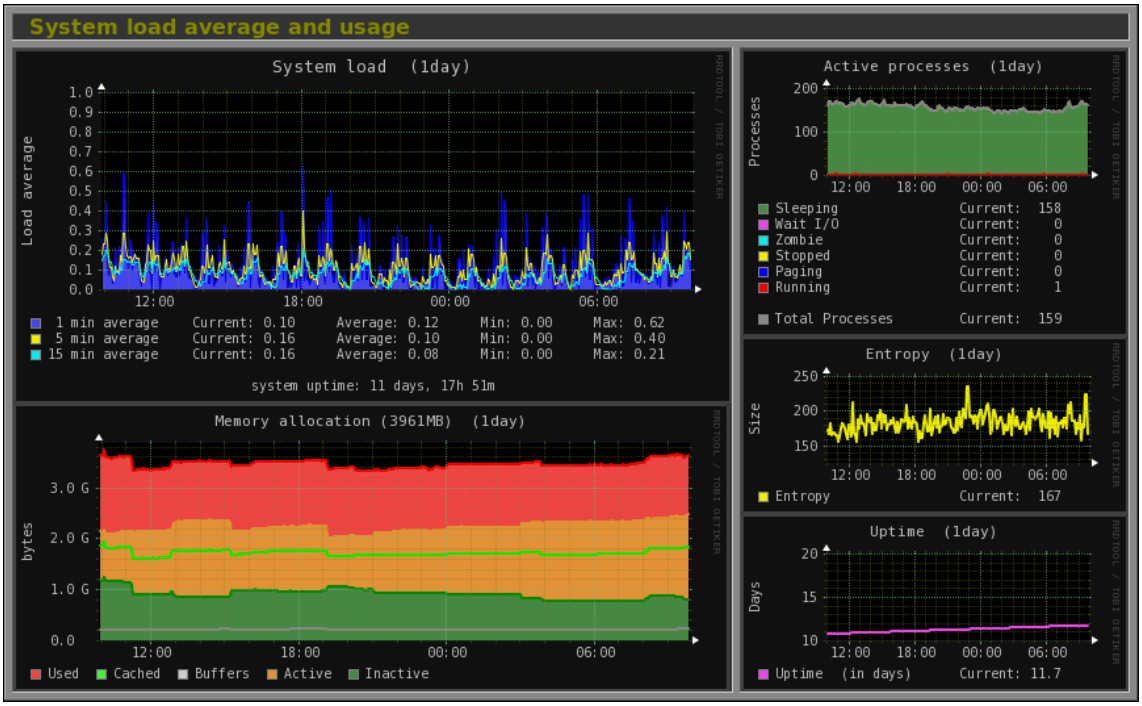
3.3 Monitorix

Monitorix je bezplatný, open-source, nenáročný nástroj pro sledování systému, který slouží k monitorování co nejvíce služeb a systémových prostředků. Byl vytvořen pro použití ve výrobních serverech Linux / UNIX, ale vzhledem k jeho jednoduchosti a malé velikosti lze použít i na vestavěná zařízení.

Skládá se převážně ze dvou programů: sběrač, nazvaný monitorix, což je démon Perl, který se spouští automaticky jako každá jiná systémová služba a CGI skript s názvem monitorix.cgi. Vzhledem k tomu, že verze Monitorix 3.0 obsahuje vlastní zabudovaný HTTP server, není potřeba používat webový server jiného výrobce.

Celý vývoj byl původně vytvořen pro sledování systémů Red Hat, Fedora a CentOS Linux, takže tento projekt byl vytvořen s ohledem na tyto typy distribuce. Dnes běží na různých distribucích GNU / Linux a dokonce i v jiných systémech UNIX, jako jsou FreeBSD, OpenBSD a NetBSD.

V současné době je v aktivním vývoji přidání nových funkcí, nových grafů a oprava chyb v pokusu nabídnout skvělý nástroj pro každodenní správu systémů. [12]



Obrázek 3.3: *Ukázka Monitorix*

3.4 Srovnání monitorovacích nástrojů

Z těchto tří nástrojů, se pro celkové potřeby řešení jeví nejlépe Monitorix. Oproti Grafaně a Kibaně nepotřebuje doinstalovávat cokoliv navíc, ať už je to webový server nebo jakákoliv součást pro funkčnost jako Elasticsearch. To jsou totiž jen další věci navíc, které do systému přinášejí více procesů, a tak můžou ovlivňovat výsledné testování. Monitorix z nich má také nejnižší nároky na operační paměť při běhu, reálný pohyb je zhruba do 80 MB.

4 Návrh řešení

Ještě před samotnou instalací a konfigurací jednotlivých nástrojů bylo zapotřebí si vytvořit návrh řešení. Tento návrh obsahuje jaké hardwarové a softwarové vybavení bylo použito, rozdělení fyzických zdrojů vzhledem k hardwaru a návrh služeb, které na serveru budou provozovány v kontejnerech.

4.1 Hardwarové vybavení

Jako základ pro server byl vybrán desktop Dell Optiplex 760 s následující specifikací:

- Základní deska: OEM pro Dell Optiplex 760 - E93839 GA0403
- Procesor: Intel Core 2 Duo E8400 3,0 GHz
- Operační paměť: 3788 MB
- Pevný disk: 1x 160GB
- Zdroj: SFF 255W

4.2 Softwarové vybavení

Jakožto operační systém byl dle zadání zvolen Linux, na němž poběží serverové služby. Konkrétně distribuci Ubuntu 16.04 „Xenial Xerus“ určenou pro servery. Jedná se o LTS verzi s podporu do roku 2021.

4.3 Návrh služeb

Služby, které jsem se rozhodl na serveru provozovat, budou dva webové servery od Nginx. Nginx je open source software pro webové servery, reverzní proxy, ukládání do mezipaměti, vyvažování zátěže, streaming médií a další. Začalo to jako webový server navržený pro maximální výkon a stabilitu. Kromě možností HTTP serveru může Nginx fungovat také jako proxy server pro e-mail (IMAP, POP3 a SMTP) a reverzní proxy a balancer zatížení pro servery HTTP, TCP a UDP.

4.4 Návrh rozdělení fyzických zdrojů

Nejvhodnějším rozdělením se jeví pro každý webový server vytvořit svůj vlastní kontejner s dostatkem operační paměti a jádrem procesoru. To jsou nejdůležitější zdroje, na kterých tyto služby závisí. Musíme ale myslet i na nějakou paměť pro hostitelský operační systém a také pro monitorovací nástroj atd. Rozdělení paměti je tedy utvořeno z celkové dostupné paměti bez těchto potřebných možností pro výše zmíněné. Samozřejmě kontejnery jako takové budou limitovány pouze maximální velikostí paměti, kterou mohou použít. To ale neznamená, že ji budou při procesu používat vždy a všechnu dostupnou.

Stručně řečeno, toto rozdělení je základní a bylo případně dále upravováno podle výkonnostních testů, aby byly co možná nejefektivnější. Přehled je tedy vytvořen v tabulce 1.1.

Tabulka 1.1: *Rozdělení fyzických zdrojů*

Služba	Operační paměť [MB]	Počet jader procesoru
WWW Nginx	512	1
WWW Nginx	512	1

4.5 Způsoby síťování při virtualizaci

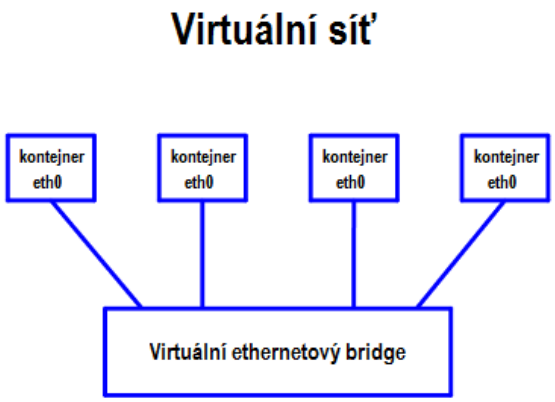
Důležitým krokem při implementaci jednoho z nástrojů pro kontejnerovou virtualizaci je výběr vhodného způsobu síťování pro aplikaci. Nejvhodnějším způsobem je využití bridge. Nástroje, které totiž budeme používat, tento způsob síťování vytvářejí automaticky i s případným rozšířením. Ať už přímo jako volbu při zadávání instalace, nebo automaticky včetně propojení kontejnerů. Pro příklad si zde ale ukážeme i další způsoby síťování, které je možno použít. [13][14][15]

4.5.1 Žádný

Při použití typu „žádný“ síťový kontejner jednoduše sdílí všechny síťové prostředky s hostitelem. Tento typ nastavení sítě je vhodný pro jednoduché testy nebo pro spuštění (grafické) aplikace v sandboxu. Vzhledem k tomu, že porty UDP / TCP jsou sdíleny s hostitelem, přidělení naslouchajících portů v kontejneru může být v rozporu s těmito přiděleními v hostiteli a naopak. Pro většinu aplikací je typ sítě typu „žádný“ nevhodný, protože kontejner má neomezený přístup k síťovým zdrojům hostitele. Není nic, co by zastavilo kontejner z ohrožení konfigurace hostitelské sítě.

4.5.2 Virtuální síť Ethernet

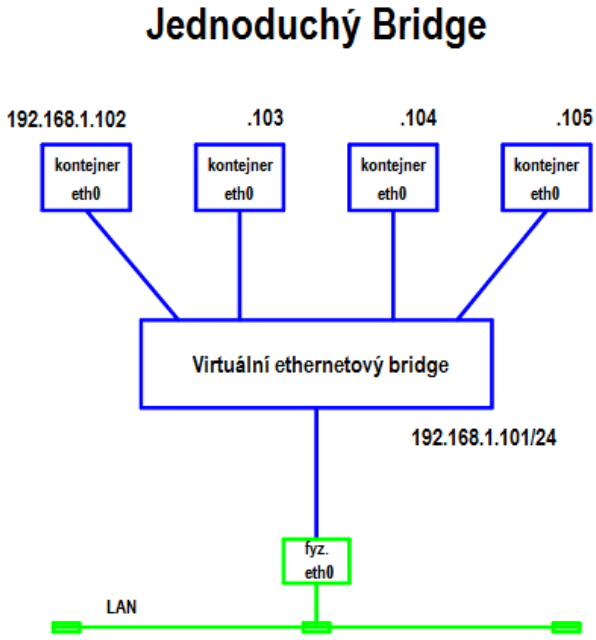
Pro většinu aplikací obsahuje nastavení sítě nástroje pro kontejnerovou virtualizaci vytvoření virtuální sítě Ethernet se středem virtuálního „přemosťovacího“(bridge) zařízení Ethernet. Každý kontejner je propojen s tímto ethernetovým bridge zařízením prostřednictvím virtuální dvojice ethernetových zařízení, z nichž jedna polovina je umístěna uvnitř kontejneru a druhá polovina je mimo kontejner. Vnější polovina těchto virtuálních ethernetových zařízení je viditelná v konfiguraci sítě hosts (ifconfig). Existují pak dva způsoby, jak propojit tuto virtuální síť s vnějším světem, a to pomocí přemostění (bridging) a směrování (NAT).



Obrázek 4.1: *Schéma virtuální sítě Ethernet*

4.5.3 Jednoduchý Bridge

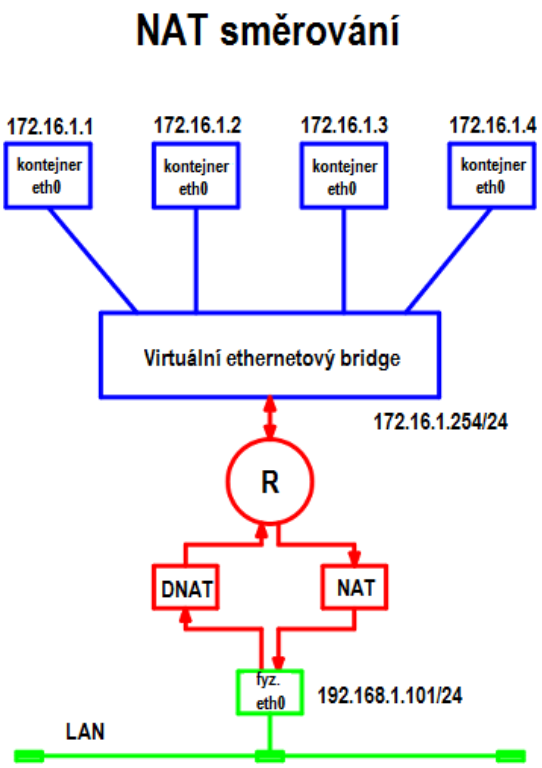
Ve způsobu „Bridge“ je virtuální ethernetový bridge přímo připojen k fyzickému síťovému rozhraní hostitele. Vzhledem k tomu, že fyzické síťové rozhraní musí procházet všemi ethernetovými rámci na odpovídajících virtuálních rozhraních kontejnerů Ethernet, musí pracovat v „promiskuitním režimu“. Adresa IP hostitele je nakonfigurována na virtuálním bridge Ethernetu.



Obrázek 4.2: *Schéma jednoduchého Bridge*

4.5.4 NAT směrování

Ve způsobu „NAT směrování“ (NAT routing) virtuální ethernetový bridge provozuje vlastní (soukromou) síť IP. Veškerý přenos do / z vnějšího světa je směrován linuxovým jádrem (kernelem). Vzhledem k tomu, že ve většině případů nechceme jednat s novou IP sítí, pravidla Linux „iptables“ NAT skryjí virtuální síť nástroje pro kontejnerovou virtualizaci před vnějším světem.



Obrázek 4.3: *Schéma směrování NAT*

4.5.5 Macvlan

Některé aplikace, zejména starší aplikace nebo aplikace, které monitorují síťovou komunikaci, očekávají, že budou přímo připojeny k fyzické síti. V tomto typu situace lze použít ovladač sítě macvlan v nástroji pro kontejnerovou virtualizaci k přiřazení adresy MAC ke každému kontejneru virtuální síťové rozhraní, takže se zdá být fyzické síťové rozhraní přímo připojené k fyzické síti. V takovém případě je nutno určit fyzické rozhraní na hostiteli, které chcete použít pro Macvlan, stejně jako podsítě a bránu Macvlan.

4.5.6 Phys

Tento typ je ze všech nejméně komplikovaný. Jednoduše spočívá v tom, že již existující rozhraní určené nástrojem pro kontejnerovou virtualizaci je přiřazeno kontejneru.

5 Instalace a konfigurace vybraných nástrojů

Nyní přistoupíme k realizaci řešení, což zahrnuje instalaci a konfiguraci vybraných nástrojů. Každý z nástrojů bude popisován zvlášť a to jak jeho postupná instalace včetně příkazů, tak následná konfigurace jednotlivých komponentů a služeb. Nakonec jednoduchými způsoby ověříme funkčnost a běh kontejnerů včetně funkčnosti virtuálních serverů s jejich službami.

Na fyzický hardware jsem tedy nainstaloval operační systém. Popis instalace operačního systému sice není předmětem této práce, ale je vhodné pro jistotu zmínit, že při instalaci není zapotřebí provést nějaké speciální nastavení. Tudíž pro tyto účely plně postačuje klasická instalace. Pro co nejlepší optimální výsledky jsem při instalaci každého dalšího nástroje provedl čistou instalaci operačního systému Ubuntu, aby případné výkonnostní testy nebyly nějakým způsobem ovlivňovány.

5.1 Virtualizace pomocí nástroje Docker

Docker je dostupný ve dvou různých edicích. Komunitní edici tak zvané Community Edition (CE) a edici pro podniky známou jako Enterprise Edition (EE).

Komunitní edice Dockeru (CE) je ideální pro vývojáře a malé týmy, kteří chtějí začít s aplikací Docker a experimentovat s aplikacemi založenými na kontejnerech. Edice Dockeru pro podniky (EE) je určena pro podnikový vývoj a týmy IT, které vytvářejí, dodávají a provozují kritické aplikace v produkčním měřítku.

Logickou volbou tedy byla Komunitní edice. [16][17][18]

5.1.1 Nastavení repozitáře

Před instalací aplikace Docker CE poprvé do nového hostitelského počítače je třeba nastavit repozitář Dockeru. Poté můžeme nainstalovat a aktualizovat Docker z repozitáře.

Nejprve je potřeba aktualizovat index balíčku apt:

```
# sudo apt-get update
```

Následuje příkaz pro instalaci balíčků, které umožňují apt použít repozitář přes http:

```
# sudo apt-get install \
apt-transport-https \
ca-certificates \
curl \
software-properties-common
```

Nyní přidáme oficiální GPG klíč Dockeru pomocí následujícího příkazu:

```
# curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo
apt-key add -
```

Teď stačí již jen přidat repozitář Dockeru do zdrojů apt:

```
# sudo add-apt-repository "deb [arch=amd64]
https://download.docker.com/linux/ubuntu $(lsb_release -cs)
stable"
```

5.1.2 Instalace Docker CE

Opět je potřeba aktualizovat index balíčku apt:

```
# sudo apt-get update
```

Nakonec tímto příkazem už lze nainstalovat samotný Docker:

```
# sudo apt-get install docker-ce
```

Docker je nyní nainstalován, bridge automaticky vytvořen, démon spuštěn a zároveň je povoleno spouštění procesu. Zda je spuštěn, můžeme zkontrolovat jednoduchým zadáním příkazu *sudo service docker status*. Výstup nám ukáže, že je služba aktivní a běží.

```
root@server01:/home/user1# service docker status
● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
   Active: active (running) since Sun 2018-04-01 14:11:32 CEST; 51s ago
     Docs: https://docs.docker.com
 Main PID: 25451 (dockerd)
   CGroup: /system.slice/docker.service
           ├─25451 /usr/bin/dockerd -H fd://
           └─25458 docker-containerd --config /var/run/docker/containerd/containerd.toml
```

Obrázek 5.1: *Výstup příkazu service docker status*

5.1.3 Vytvoření docker compose souboru

Soubor docker-compose.yml je soubor yaml, který definuje, jak se mají kontejnery Docker chovat. Nginx je znám jako skvělé reverzní proxy řešení. Využijeme tedy toho a vytvoříme tak kontejnery pro webové servery a spojení mezi nimi bude pomocí kontejneru Nginx reverzní proxy.

Pro efektivnější funkčnost použijeme dva základní Docker obrazy přímo z Docker Hubu a jeden vlastní obraz, přičemž vlastní obraz je naše reverzní proxy.

Vytvoříme tedy soubor docker-compose.yml následujícím způsobem:

```
 GNU nano 2.5.3          File: docker-compose.yml

version: '3'

services:
    reverseproxy:
        image: reverseproxy
        ports:
            - 80:80
            - 8081:8081
        restart: always

    nginx1:
        depends_on:
            - reverseproxy
        image: nginx:alpine
        restart: always
        resources:
        limits:
          cpus: "0-1"
          memory: 512MB

    nginx2:
        depends_on:
            - reverseproxy
        image: nginx:alpine
        restart: always
        resources:
        limits:
          cpus: "1-2"
          memory: 512MB
```

Obrázek 5.2: *Obsah souboru docker-compose.yml*

Služba reverseproxy použije náš vlastní obraz, který vytvoříme v dalším kroku. Služby nginx1 a nxing2 budou používat každá svůj obraz a jsou závislé na dostupnosti služby reverseproxy. Pouze porty v službě reverseproxy jsou vystaveny hostitelskému počítači. To je chápáno jako dobře, protože to znamená, že hostitel nebude moci komunikovat přímo s žádnou z exponovaných služeb našich ostatních kontejnerů. V produkčním prostředí je samozřejmě pravděpodobné nutnost chtít, aby reverzní proxy používaly porty 80 a 443, ale protože děláme všechno lokálně bez jména serverů, musíme rozlišit naše webové služby podle portů. První bude tedy používat port 80 a druhá port 8081.

Nakonec jsme webovým službám v kontejnerech nastavili omezující použití operační paměti a procesoru. Paměť se zadává v megabytech a u procesoru nastavení „0-1“ by nám mělo zapříčinit využití pouze prvního jádra, taktéž nastavení „1-2“ by mělo využívat pouze druhého jádra. Pro kontejner reverseproxy jsme nic takové nenastavili z toho důvodu, aby běžel ve výchozím stavu podobně jako ostatní procesy a neovlivňoval tak původní koncept.

5.1.4 Vytvoření konfiguračních souborů

Náš vlastní obraz představující reverzní proxy potřebuje k běhu soubor Dockerfile a opět nás vlastní Nginx konfigurační soubor. Nazveme jej nginx.conf.

Otevřeme Dockerfile a vložíme do něj následující:

```
  GNU nano 2.5.3                File: Dockerfile

FROM nginx:alpine

COPY nginx.conf /etc/nginx/nginx.conf
```

Obrázek 5.3: *Obsah souboru Dockerfile*

Tento vlastní obraz bude mít základní obraz Alpine se systémem Nginx. Velikostně je mnohem menší a zároveň méně náročný než klasický obraz Nginx, což je jedna z výhod Dockeru co se týče vytvořených obrazů. Během procesu sestavení bude náš konfigurační soubor zkopírován do obrazu.

Teď si vytvoříme náš výše zmíněný soubor s názvem nginx.conf a vložíme do něj následující:

```
  GNU nano 2.5.3              File: nginx.conf

worker_processes 1;

events { worker_connections 1024; }

http {

    sendfile on;

    upstream docker-nginx1 {
        server nginx1:80;
    }

    upstream docker-nginx2 {
        server nginx2:80;
    }

    server {
        listen 80;

        location / {
            proxy_pass         http://docker-nginx1;
            proxy_redirect     off;
            proxy_set_header   Host $host;
            proxy_set_header   X-Real-IP $remote_addr;
            proxy_set_header   X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header   X-Forwarded-Host $server_name;
        }
    }

    server {
        listen 8081;

        location / {
            proxy_pass         http://docker-nginx2;
            proxy_redirect     off;
            proxy_set_header   Host $host;
            proxy_set_header   X-Real-IP $remote_addr;
            proxy_set_header   X-Forwarded-For $proxy_add_x_forwarded_for;
            proxy_set_header   X-Forwarded-Host $server_name;
        }
    }

}
```

Obrázek 5.4: *Obsah konfiguračního souboru nginx.conf*

V tomto souboru máme deklarovány dva upstreamy, protože máme dvě webové aplikace. Server uvnitř každého z upstreamů představuje místo, kde lze nalézt jednotlivé aplikace. Název musí odpovídat názvu služby v souboru docker-compose.yml. Ve výchozím

nastavení se na portu 80 vysílají webové servery Nginx1 a Nginx2. Po definování upstream serverů jsme museli Nginxu říct, jak naslouchat na portu a jak reagovat na požadavky.

Pokud se pokusíme přistupovat k hostitelskému stroji prostřednictvím portu, Nginx bude sloužit jako reverzní proxy a postoupí cokoli v definici *proxy_pass*.

5.1.5 Vytvoření kontejnerů

Předtím, než můžeme spustit naše kontejnery, musíme vytvořit náš reverseproxy obraz. To lze snadno provést provedením následujícího příkazu:

```
# docker build -t reverseproxy
/cesta_k_souboru_dockerfile_a_nginx.conf/
```

Soubor docker-compose.yml očekává obraz nazvaný reverseproxy, takže tím to teď vytváříme. Soubor Dockerfile a nginx.conf musí existovat ve stejném umístění.

Nyní už konečně jen zakončit příkazem, který vše poskládá a vytvoří:

```
# docker-compose up -d
```

```
root@server01:/home/user1# docker-compose up -d
Creating user1_reverseproxy_1 ... done
Creating user1_nginx2_1       ... done
Creating user1_nginx1_1       ... done
```

Obrázek 5.5: *Výstup příkazu docker-compose up -d*

Po dokončení máme tři nasazené kontejnery se zároveň automaticky vytvořenými rozhraními a propojení s bridgem.

5.1.6 Ověření a testování virtuálních serverů

Nejdříve si můžeme pomocí příkazu # *docker ps -a* zkontrolovat, zda jsou všechny kontejnery opravdu vytvořeny a spuštěny. Ve výpisu jsou pro nás důležité tři id kontejnerů a ve statusu hodnota „Up“. Můžeme také vidět nastavení portů.

```
root@server01:/home/user1# docker ps -a
CONTAINER ID        IMAGE               COMMAND                  CREATED             STATUS              PORTS
                    NAMES
49046d02c8f4        nginx:alpine        "nginx -g 'daemon of…"   2 hours ago         Up 2 hours          80/tcp
                    user1_nginx1_1
b6db6de7b3ab        nginx:alpine        "nginx -g 'daemon of…"   2 hours ago         Up 2 hours          80/tcp
                    user1_nginx2_1
73fdc8086093        reverseproxy        "nginx -g 'daemon of…"   2 hours ago         Up 2 hours          0.0.0.0:80->80/tcp,
 0.0.0.0:8081->8081/tcp   user1_reverseproxy_1
```

Obrázek 5.6: *Výstup příkazu docker ps -a*

Nadále můžeme zkontrolovat přímo funkčnost webových serverů zadáním IP adresy serveru a příslušného portu do svého webového prohlížeče v lokálním počítači, kde se zobrazí stránka s uvítací větou "Welcome to nginx!", která signalizuje správnou funkčnost.

5.2 Virtualizace pomocí nástroje LXC

Ve většině případů najdeme aktuální verze LXC k dispozici pro distribuci v systému Linux. Buď přímo v úložišti balíčků nebo přes nějaký backport kanál. [19][20][21]

5.2.1 Instalace LXC

Nejdříve je tedy potřeba aktualizovat index balíčku apt:

```
# sudo apt-get update
```

Následujíc zadáním komplexního příkazu pro instalaci LXC i se šablonami a potřebnými balíčky pro budoucí běh kontejnerů:

```
# sudo apt-get install lxc lxc-templates wget bridge-utils
```

Po zadání příkazu můžeme sledovat veškerá nastavení, která se během instalace zavádí, jako konfigurace kernelu, cgroups, vytvoření bridge atd. Pro ověření úspěšné instalace použijeme příkaz *sudo service lxc status*. Výstup nám ukáže, že je služba aktivní a běží.

```
root@server01:/home/user1# service lxc status
● lxc.service - LXC Container Initialization and Autoboot Code
   Loaded: loaded (/lib/systemd/system/lxc.service; enabled; vendor preset: enabled)
   Active: active (exited) since Thu 2018-04-12 15:16:24 CEST; 2min 3s ago
     Docs: man:lxc-autostart
           man:lxc
  Process: 1239 ExecStart=/usr/lib/x86_64-linux-gnu/lxc/lxc-containers start (code=exited, status=0/SUCCESS)
  Process: 1228 ExecStartPre=/usr/lib/x86_64-linux-gnu/lxc/lxc-apparmor-load (code=exited, status=0/SUCCESS)
 Main PID: 1239 (code=exited, status=0/SUCCESS)
    Tasks: 0
   Memory: 0B
      CPU: 0
   CGroup: /system.slice/lxc.service

Apr 12 15:16:24 server01 systemd[1]: Starting LXC Container Initialization and Autoboot Code...
Apr 12 15:16:24 server01 systemd[1]: Started LXC Container Initialization and Autoboot Code.
```

Obrázek 5.7: *Výstup příkazu service lxc status*

5.2.2 Vytvoření kontejneru

Jelikož jsme při instalaci zadali i parametr pro šablony, LXC nám nabízí připravené šablony pro snadnou instalaci kontejnerů pro Linux. Tyto šablony se obvykle nacházejí v /usr/share/lxc/templates.

Pro vypsání je potom nutnost zadat příkaz:

```
# sudo ls /usr/share/lx/templates/
```

```
root@server01:/home/user1# ls /usr/share/lxc/templates
lxc-alpine      lxc-centos     lxc-fedora        lxc-oracle      lxc-sshd
lxc-altlinux    lxc-cirros     lxc-gentoo        lxc-plamo       lxc-ubuntu
lxc-archlinux   lxc-debian     lxc-openmandriva  lxc-slackware   lxc-ubuntu-cloud
lxc-busybox     lxc-download   lxc-opensuse      lxc-sparclinux
```

Obrázek 5.8: *Výpis šablon lxc*

V tomto výpisu vidíme všechny šablony, které můžeme použít. Nyní vytvoříme kontejner pomocí šablony lxc-ubuntu, která nám zajistí poslední verzi Ubuntu vztaženou k našemu hostitelskému operačnímu systému. Jedná se tedy o 16.04.4 LTS.

Je nutno provést následujícím příkazem:

```
# sudo lxc-create -n nginx1 -t ubuntu
```

Kde parametr -n udává jméno kontejneru, v našem případě nginx1. Parametr -t udává výše zmíněnou šablonu. Po potvrzení tohoto příkazu, se vytvoří kontejner společně s napojením na bridge, do kterého se začne stahovat a instalovat daný systém Ubuntu. Na konci výpisu se zobrazí, že pro daný systém se automaticky vytvořil defaultní uživatelský účet s názvem *ubuntu* a heslo je taktéž *ubuntu*.

Jakmile je kontejner vytvořen, spustíme jej pomocí následujícího příkazu, kde parametr -d udává spuštění kontejneru na pozadí:

```
# sudo lxc-start -n nginx1 -d
```

Abychom se dostali do konzole spuštěného kontejneru, použijeme příkaz:

```
#sudo lxc-console -n nginx1
```

Následuje výpis a požadavek na přihlášení do systému, tak jako bychom se přihlašovali normálně do hostitelského. Zde použijeme výše zmíněné přihlašovací údaje.

```
root@server01:/home/user1# lxc-console -n nginx1

Connected to tty 1
                Type <Ctrl+a q> to exit the console, <Ctrl+a Ctrl+a> to enter
Ctrl+a itself

Ubuntu 16.04.4 LTS nginx1 pts/0

nginx1 login: ubuntu
Password:
Last login: Thu Apr 12 12:23:23 UTC 2018 on pts/0
Welcome to Ubuntu 16.04.4 LTS (GNU/Linux 4.4.0-87-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage
```

Obrázek 5.9: *Přihlášování do kontejneru*

Nyní se nacházíme uvnitř kontejneru, a tak můžeme provést veškerou prací stejně jako na Linuxových serverech. Přistoupíme k instalaci webového serveru. K aktualizaci indexu balíčku apt a instalaci Nginx uvnitř kontejneru použijeme následujících příkazů:

```
# sudo apt-get update
```

```
# sudo apt-get install nginx
```

Pro pozdější kontrolu rozdělení kontejnerů upravíme výchozí webovou stránku pro tento web. Z textu bude vyplívat, že daný web je provozován v kontejneru nginx1. Otevřeme soubor zadáním příkazu:

```
# sudo nano /var/www/html/index.nginx-debian.html
```

Otevřený soubor upravíme podle vzoru viz Obrázek 5.10, tak aby výsledkem byl srozumitelný výstup, ze kterého je možné identifikovat použitý kontejner.

```
  GNU nano 2.5.3            File: /var/www/html/index.nginx-debian.html

<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx - LXC Kontejner1 !</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx - LXC Kontejner1!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Obrázek 5.10: *Úprava výchozí webové stránky*

Po dokončení můžeme zavřít. Pro odhlášení z konzole kontejneru stiskneme klávesy „Ctrl+a q“. Teď se už nacházíme zpátky v terminálu hostitelského počítače.

5.2.3 Klonování kontejnerů

LXC nabízí funkci klonování kontejneru z existujícího kontejneru. Vzhledem k našemu řešení je to velice výhodné, protože takto nemusíme znovu provádět stahování a instalaci systému a instalaci webového serveru.

Předtím než ale můžeme začít, musíme kontejner zastavit. To provedeme pomocí příkazu:

```
# sudo lxc-stop -n nginx1
```

Pro vytvoření kopie našeho kontejneru nginx1 je nutno provést příkaz:

```
# sudo lxc-copy -n nginx1 -N nginx2
```

Za parametrem -N jsme zadali nové výchozí jméno klonu původního kontejneru. Pro ověření, že se kontejner vytvořil, můžeme použít příkaz *sudo lxc-ls*, kde se výpisu objeví obě zmíněná jména kontejnerů.

Když je kontejner s názvem nginx2 vytvořen, je nutné jej již známým příkazem spustit a přihlásit se do něj:

```
# sudo lxc-start -n nginx2 -d
# sudo lxc-console -n nginx2
```

Následně provedeme podobnou editaci výchozí webové stránky v již nainstalovaném nginxu, kde je vlastně nutnost pozměnit číslovku jedna za číslovku dva. Jakmile máme provedeno, odhlásíme se z kontejneru.

5.2.4 Editace konfiguračního souboru

Jakmile máme oba kontejnery vytvořeny, ještě jim chybí nastavit omezení pro využívání fyzických zdrojů. Musíme tedy upravit konfigurační soubory. Do těchto souborů je potřeba přidat několik nových řádků, jimiž omezíme paměť na 512MB a přiřadíme jádro. Zde jsou opět daná jádra procesoru detekována jako procesor 0 a procesor 1. Jejich užití je definováno příkazem, který lze nalézt v obecné dokumentaci.

Konfigurační soubor se nachází ve /var/lib/lxc/jméno_kontejneru/config. Editaci a provedení následujících změn tak provedeme příkazem:

```
# sudo nano /var/lib/lxc/nginx1/config
```

```
  GNU nano 2.5.3          File: /var/lib/lxc/nginx1/config

# Template used to create this container: /usr/share/lxc/templates/lxc-ubuntu
# Parameters passed to the template:
# Template script checksum (SHA-1): 4d7c613c3c0a0efef4b23917f44888df507e662b
# For additional config options, please look at lxc.container.conf(5)

# Uncomment the following line to support nesting containers:
#lxc.include = /usr/share/lxc/config/nesting.conf
# (Be aware this has security implications)


# Common configuration
lxc.include = /usr/share/lxc/config/ubuntu.common.conf

# Container specific configuration
lxc.rootfs = /var/lib/lxc/nginx1/rootfs
lxc.rootfs.backend = dir
lxc.utsname = nginx1
lxc.arch = amd64

# Network configuration
lxc.network.type = veth
lxc.network.link = lxcbr0
lxc.network.flags = up
lxc.network.hwaddr = 00:16:3e:74:1b:e3

# Memory set
lxc.cgroup.memory.limit_in_bytes = 512M

# Core set
lxc.cgroup.cpuset.cpus = 0
```

Obrázek 5.11: *Obsah konfiguračního souboru /lxc/nginx1/config*

Velikost paměti se zadává v bytech, proto musíme do souboru napsat hodnotu 512M. Po uložení je toto nastavení provedeno pouze pro první kontejner. Pro druhý se musí provést to samé, až na parametr pro nastavení jádra, kde uvedeme hodnotu 1.

5.2.5 Ověření a testování virtuálních serverů

Zde si můžeme pro ověření funkčnosti webový serverů nechat vypsat list všech vytvořených kontejnerů. Lze to provést příkazem *sudo lxc-ls -f*. Ve výpisu můžeme vidět důležitou položku statusu, konkrétně *running*, která dává najevo běžící kontejner. Součástí jsou také konkrétní IP adresy daných kontejnerů.

```
root@server01:/home/user1# lxc-ls -f
NAME   STATE   AUTOSTART GROUPS IPV4       IPV6
nginx1 RUNNING 0         -      10.0.3.35  -
nginx2 RUNNING 0         -      10.0.3.37  -
```

Obrázek 5.12: *Výstup příkazu lxc-ls -f*

Nyní pro změnu použijeme k otestování příkaz curl, jenž ověří funkčnost webových serverů v jednotlivých kontejnerech. K tomu právě potřebujeme výše zmíněné IP adresy kontejnerů. Curl je nástroj pro přenos dat z nebo na server pomocí podporovaného protokolu http. V podstatě jej můžeme použít pro stažení obsahu z internetu, v našem případě obsahu vytvořených webových serverů v kontejnerech, a tak potvrzení jejich funkčnosti podle výše provedené přípravy.

Pro potvrzení použijeme tento příkaz:

```
# curl http://IP_adresa_kontejneru/
```

```
root@server01:/home/user1# curl http://10.0.3.35/
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx - LXC Kontejner1 !</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx - LXC Kontejner1!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Obrázek 5.13: *Výstup příkazu curl pro první webový server kontejneru LXC*

```
root@server01:/home/user1# curl http://10.0.3.37/
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx - LXC Kontejner2 !</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx - LXC Kontejner2 !</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Obrázek 5.14: *Výstup příkazu curl pro druhý webový server kontejneru LXC*

5.3 Virtualizace pomocí nástroje LXD

S ohledem na to, že používáme serverový operační systém Ubuntu, tak LXD je již předinstalován a je tak součástí systému. [22][23][24]

5.3.1 Instalace LXD

Pokud bychom však potřebovali nainstalovat LXD na klasický Ubuntu systém nebo na nějaký jiný, lze to udělat pomocí příkazů:

```
# sudo apt update
# sudo apt install lxd
```

Poté bude LXD nainstalováno a dostáváme se tak do stejného bodu pro pokračování instalace.

5.3.2 Přidání uživatele do skupiny lxd

Je vhodné se nejdříve přihlásit do serveru pomocí uživatelského účtu bez oprávnění root. Pomocí tohoto účtu budeme provádět všechny úkoly, co se týče správy kontejnerů. Aby to fungovalo, musíme nejprve přidat tohoto uživatele do skupiny lxd.

Následuje tedy příkaz:

```
# sudo usermod --append --groups lxd "uživatelský_účet"
```

5.3.3 Konfigurace úložiště

LXD je třeba správně nakonfigurovat, než jej budeme moci používat. Nejdůležitějším konfiguračním rozhodnutím je typ úložiště pro ukládání kontejnerů. Doporučené backend úložiště pro LXD je souborový systém ZFS uložený buď v předem určeném souboru, nebo pomocí blokového úložiště. Pro použití podpory ZFS v LXD, je nutno nainstalovat balíček *zfsutils-linux*.

Nejprve je potřeba aktualizovat index balíčku apt:

```
# sudo apt-get update
```

Následuje příkaz pro instalaci zfs:

```
# sudo apt-get install zfsutils-linux
```

S touto instalací jsme připraveni inicializovat LXD. Během inicializace jsme vyzváni k zadání podrobností pro backend úložiště ZFS. Pro naše potřeby je vhodnější variantou použít předem uložený soubor, protože používáme jen jeden pevný disk s jedním oddílem.

Nejprve je potřeba spustit následující inicializační proces LXD příkazem:

```
# lxd init
```

Načež jsme požádání o poskytnutí několika informací, jak lze vidět na obrázku 5.15. Vybíráme všechny výchozí hodnoty, včetně doporučené velikosti pro předem přidělený soubor nazván jako „loop device“.

```
root@server01:/home/user1# lxd init
Do you want to configure a new storage pool (yes/no) [default=yes]? yes
Name of the storage backend to use (dir or zfs) [default=zfs]: zfs
Create a new ZFS pool (yes/no) [default=yes]? yes
Name of the new ZFS pool or dataset [default=lxd]: lxd
Would you like to use an existing block device (yes/no) [default=no]? no
Size in GB of the new loop device (1GB minimum) [default=28]: 28
Would you like LXD to be available over the network (yes/no) [default=no]? no
Do you want to configure the LXD bridge (yes/no) [default=yes]? yes
Warning: Stopping lxd.service, but it can still be activated by:
  lxd.socket
^[LXD has been successfully configured.
```

Obrázek 5.15: *Průvodní nastavení nástroje LXD*

Navrhovaná velikost se automaticky vypočítává z dostupného místa na disku našeho serveru.

5.3.4 Konfigurace sítě

Jakmile je předchozí proces dokončen, jsme vyzváni ke konfiguraci sítě. Inicializační proces nám představí řadu kontextových oken, které nám umožní konfigurovat síťový bridge pro kontejnery, aby mohly získat soukromé IP adresy a vzájemně komunikovat mezi sebou.

Použijeme výchozí hodnotu pro každou možnost. Během procesu je možnost nastavit i vlastní subnet, popřípadě pozměnit název rozhraní bridge. Po dokončení instalace je vše připraveno pro vytvoření kontejnerů.

5.3.5 Vytvoření kontejnerů

Úspěšně jsme nakonfigurovali LXD. Určili jsme umístění úložiště a nakonfigurovali jsme výchozí síť pro všechny nově vytvořené kontejnery. Nyní už můžeme přistoupit k tvorbě kontejnerů samotných, přičemž ve většině příkazů se začíná pomocí „lxc“. Nejprve si ale ověříme funkčnost nástroje jako takového.

Pro ověření funkčnosti je nejvhodnější příkaz, který uvádí dostupné nainstalované kontejnery:

```
# lxc list
```

A uvidíme tak následující výstup:

```
root@server01:/home/user1# lxc list
Generating a client certificate. This may take a minute...
If this is your first time using LXD, you should also run: sudo lxd init
To start your first container, try: lxc launch ubuntu:16.04

+------+-------+------+------+------+-----------+
| NAME | STATE | IPV4 | IPV6 | TYPE | SNAPSHOTS |
+------+-------+------+------+------+-----------+
```

Obrázek 5.16: *Výstup říkazu lxc list*

Protože je to poprvé, kdy příkaz lxc komunikuje s hypervisorem LXD, výstup nám oznamuje, že příkaz automaticky vytvořil klientský certifikát pro bezpečnou komunikaci s LXD. Poté zobrazuje informace o tom, jak spustit kontejner. Nakonec příkaz zobrazí prázdný seznam kontejnerů, což je pochopitelné, protože jsme zatím žádný nevytvořili.

Teď už ale k tvorbě samotných kontejnerů. Pro ty je potřeba použít příkazy:

```
# lxc launch ubuntu:x nginx1
# lxc launch ubuntu:x nginx2
```

Přikaz *lxc launch* je k vytvoření a spuštění kontejnerů Ubuntu 16.04 (ubuntu:x) s názvy nginx1 a nginx2. To *x* v *ubuntu:x* je zkratka pro první písmeno Xenial, kódové označení pro Ubuntu 16.04. *Ubuntu:* je identifikátor pro předkonfigurované úložiště LXD obrazů. Protože je to poprvé, co jsme vytvořili kontejner, první příkaz stahuje obraz kontejneru z Internetu a ukládá jej do mezipaměti. Další je vytvořen podstatně rychleji.

Aby bylo možné s kontejnerem něco dělat, je potřeba se k němu nejdříve přihlásit. K přihlášení se používá příkaz *lxc exec*, který má název kontejneru a příkazy k jeho spuštění. Pro přistoupení k prvnímu kontejneru tedy:

```
# lxc exec nginx1 --sudo --login --user ubuntu
```

Řetězec -- označuje, že parametry příkazu lxc by se na tom místě měly zastavit a zbytek řádku se předá jako příkaz, který má být proveden uvnitř kontejneru. Příkaz *sudo --login --user ubuntu*, poskytuje přihlašovací shell pro předkonfigurovaný účet ubuntu uvnitř kontejneru. Jakmile jsme uvnitř kontejneru, shell promtp se změní na *ubuntu@nginx1: ~$* , což perfektně slouží pro přehled, zda se nacházíme uvnitř kontejneru nebo v hostitelském serveru. Tento *ubuntu* uživatel má v kontejneru předkonfigurovaný přístup sudo a může spustit sudo příkazy bez zadávání hesla. Tento shell je omezen pouze na vnitřek kontejneru. Všechno, co spouštíme v tomto shellu, zůstane v kontejneru a nemůže uniknout na hostitelský server.

Nyní přistoupíme k aktualizaci indexu balíčku apt a instalaci Nginx uvnitř kontejneru pomocí následujících příkazů:

```
ubuntu@nginx1:~$ sudo apt-get update
```

```
ubuntu@nginx1:~$ sudo apt-get install nginx
```

Opět pro pozdější kontrolu rozdělení kontejnerů upravíme výchozí webovou stránku pro tento web. Z textu bude vyplívat, že daný web je provozován v kontejneru nginx1. Otevřeme soubor zadáním příkazu:

```
ubuntu@nginx1:~$ sudo nano /var/www/html/index.nginx-debian.html
```

Stačí znovu jednoduše editovat, aby to bylo na první pohled zřejmé:

```
  GNU nano 2.5.3            File: /var/www/html/index.nginx-debian.html

<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx - LXD KONTEJNER1!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx - LXD KONTEJNER1!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Obrázek 5.17: *Úprava výchozího vzhledu webové stránky*

Po uložení je možné se už z kontejneru odhlásit následujícím příkazem:

```
ubuntu@nginx1:~$ logout
```

To samé je nyní nutné udělat i pro druhý kontejner ve stejném sledu. Přihlásit se do něj pomocí:

```
# lxc exec nginx2 --sudo --login --user ubuntu
```

Aktualizovat index balíčku apt a nainstalovat Nginx příkazy:

```
ubuntu@nginx2:~$ sudo apt-get update
ubuntu@nginx2:~$ sudo apt-get install nginx
```

Editovat výchozí stránku v tomto kontejneru s přihlédnutím na to, že se jedná o druhý kontejner příkazem:

```
ubuntu@nginx2:~$ sudo nano /var/www/html/index.nginx-debian.html
```

A pak už se stačí opět pouze odhlásit příkazem:

```
ubuntu@nginx2:~$ logout
```

Poslední věc, která je nutná kontejnerům nastavit, je přiřadit jim operační paměť a příslušné jádro procesoru.

Pro limit paměti je nutno použít následující příkaz, společně se specifikací o jaký kontejner se jedná a velikost zadávaná v megabytech:

```
# lxc config set nginx1 limits.memory 512MB
# lxc config set nginx2 limits.memory 512MB
```

Pro limit jader je nezbytné použít následující příkazy. Zde LXD detekuje jediný procesor s dvěma jádry jako *procesor 0* a *procesor 1*, je tedy nutné zadat dané parametry následujícími způsoby:

```
# lxc config set nginx1 limits.cpu 0-0
# lxc config set nginx2 limits.cpu 1-1
```

5.3.6 Ověření a testování virtuálních serverů

Zde si můžeme opět pomocí příkazu *lxc list* tak zvaně vylistovat námi všechny vytvořené kontejnery a přesvědčit se, zda jsou spuštěné. To nám dává vědět položka *running* u každého z kontejnerů. Můžeme zde i vidět, že každý z kontejnerů má přiřazenou svou vlastní IP adresu.

```
root@server01:/home/user1# lxc list
+--------+---------+----------------------+------+------------+-----------+
|  NAME  |  STATE  |         IPV4         | IPV6 |    TYPE    | SNAPSHOTS |
+--------+---------+----------------------+------+------------+-----------+
| nginx1 | RUNNING | 10.219.204.141 (eth0) |      | PERSISTENT | 0         |
+--------+---------+----------------------+------+------------+-----------+
| nginx2 | RUNNING | 10.219.204.228 (eth0) |      | PERSISTENT | 0         |
+--------+---------+----------------------+------+------------+-----------+
```

Obrázek 5.18: *Výstup příkazu lxc list pro LXD kontejnery*

Nyní použijeme k otestování znovu příkaz curl, že webové servery v kontejnerech fungují. K tomu právě potřebujeme výše zmíněné IP adresy kontejnerů.

Použijeme tedy následující příkaz:

```
# curl http://IP_adresa_kontejneru/
```

Tímto příkazem vlastně dojde k potvrzení funkčnosti, kterou jsme si připravili výše při instalaci.

```
root@server01:/home/user1# curl http://10.219.204.141/
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx - LXD KONTEJNER1!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx - LXD KONTEJNER1!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Obrázek 5.19: *Výstup příkazu curl pro první webový server kontejneru LXD*

```
root@server01:/home/user1# curl http://10.219.204.228/
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx - LXD KONTEJNER2!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx - LXD KONTEJNER2!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Obrázek 5.20: *Výstup příkazu curl pro druhý webový server kontejneru LXD*

5.4 Monitorování pomocí nástroje Monitorix

V Ubuntu se Monitorix nenachází v oficiálních repozitářích, ale můžeme přidat jeden generický od společnosti IzzySoft, který funguje jak pro Ubuntu, tak i pro Debian.

Nejdříve je tedy nutnost přidat repozitář IzzySoft do souboru /etc/apt/sources.list pomocí příkazu:

```
# sudo nano /etc/apt/sources.list
```

a na konci souboru přidat řádek:

```
deb http://apt.izzysoft.de/ubuntu generic universe
```

Poté přidat Izzysoft GPG klíč pro tento repozitář pomocí příkazu wget spuštěním:

```
# sudo wget -q http://apt.izzysoft.de/izzysoft.asc
```

Po stažení se musí přidat tento GPG klíč do apt konfigurace pomocí příkazu 'apt-key':

```
# sudo apt-key add izzysoft.asc
```

Nyní se provede update a nainstaluje Monitorix příkazy:

```
# sudo apt-get update
# sudo apt-get install monitorix
```

Současně si sám také nainstaluje všechny své potřebné komponenty.

Pro ověření funkčnosti otevřeme svůj webový prohlížeč a zadáme: http://IP_adresa_serveru:8080/monitorix

Monitorix je dodáván s výchozí konfigurací, která je plně dostačující, ale můžeme také upravit konfiguraci v souboru /etc/monitorix/monitorix.conf

Až je vše hotovo, restartujeme aplikaci Monitorix, aby se provedli změny:

```
# service monitorix restart
```

Nejlepší je ale prostudovat si dokumentaci a zjistit tak, že lze zcela jednoduchým poskládáním různých parametrů ve webovém odkaze dosáhnout našeho cíleného zobrazení grafů. [25]

Například chceme zobrazit všechny grafy, v časovém úseku jedné hodiny a s černým pozadím. Stačí to tedy dopsat za již výše zmíněný webový odkaz v následujícím formátu s definovanými parametry:

http://IP_adresa_serveru:8080/monitorix-cgi/monitorix.cgi?mode=localhost&graph=all&when=1hour&color=black

6 Výkonnostní testy

Pro testování virtuálních serverů výkonnostním testem jsem použil program *wrk*. Je to moderní nástroj pro porovnávání HTTP, který je schopen generovat významnou zátěž při běhu na více jádrovém procesoru. Kombinuje více vláknový design se systémy škálovatelných oznamovacích událostí, jako je epoll a kqueue. Volitelný skript LuaJIT může provádět generování požadavků HTTP, zpracování odezvy a vlastní reportování. [26]

Výkonnostní test tohoto nástroje byl spuštěn zároveň na oba webové servery a to na každý zvlášť. Test konkrétně spočíval ve spuštění benchmarku po dobu 20 minut, použití jednoho vlákna a udržování otevřených 300 HTTP spojení. Takovýto příkaz vypadá následovně:

```
# wrk -t1 -c300 -d20m http://IP_adresa/
```

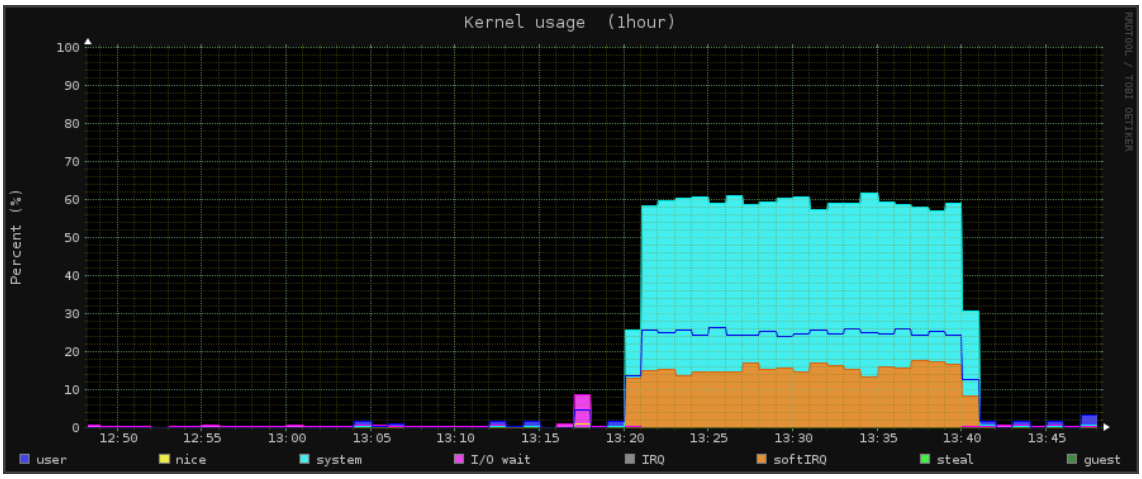
Grafy čerpání jednotlivých zdrojů jsou zachyceny pomocí nástroje Monitorix.

6.1 Docker

Test byl spuštěn v úseku 13:20 - 13:40.

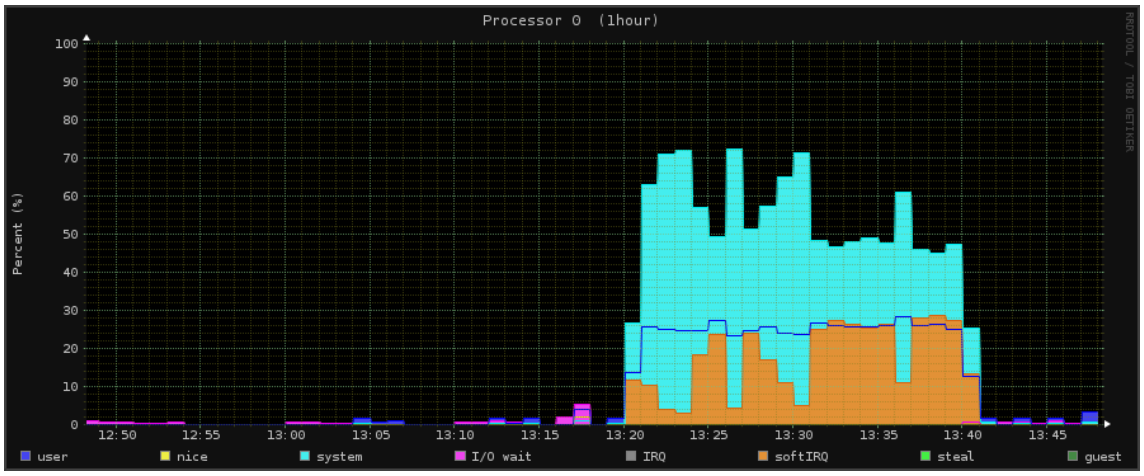
Na následujících grafech lze vidět jednotlivé využití komponent v rámci daného nástroje během spuštěného výkonnostního testu. Konkrétně lze vidět, že využití kernelu nepřesahovalo hodnotu vyšší než je zhruba 61%. U grafů využití jednotlivých jader procesoru zase můžeme vidět, že v tomto případě Docker nebyl schopný zcela dobře rozložit zátěž rovnoměrně na obě jádra. Maximální hodnoty, kterých dosáhly, byly zhruba 72% a 74%. Nadále využití operační paměti se dostalo až na zhruba 1800MB a síť v rámci udržovaných spojení dosahovala požadovaných hodnot vzhledem k zadaným testům.

6.1.1 Využití kernelu

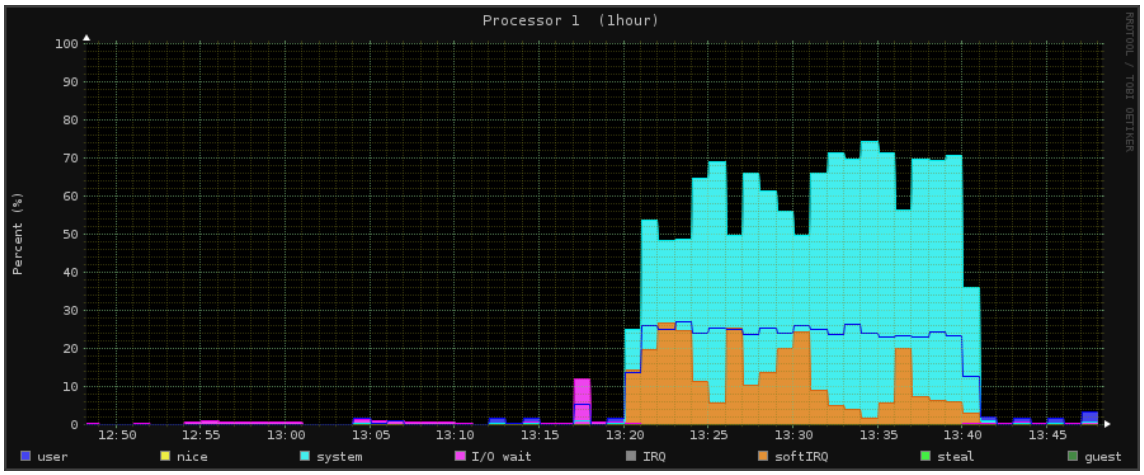


Obrázek 6.1: *Graf využití kernelu v nástroji Docker*

6.1.2 Využití jader procesoru

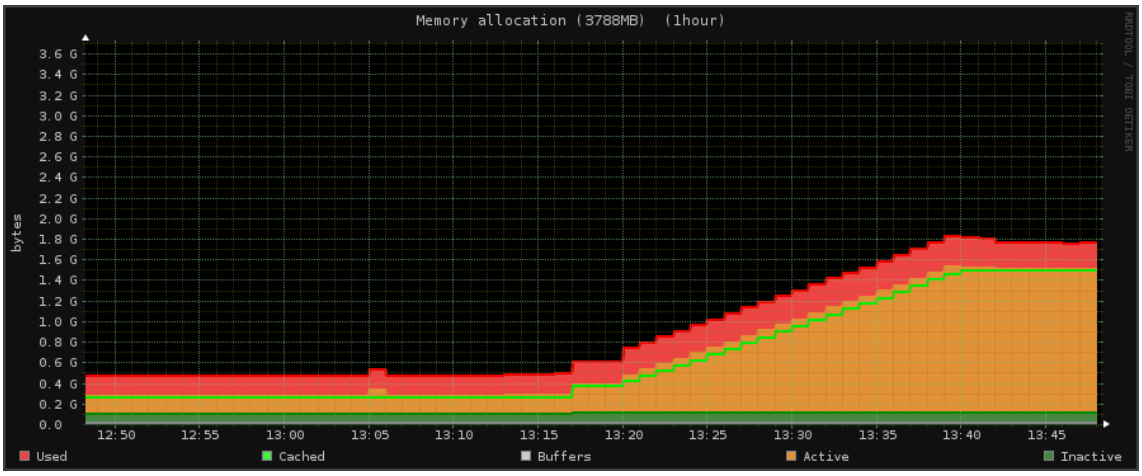


Obrázek 6.2: *Graf využití prvního jádra procesoru v nástroji Docker*



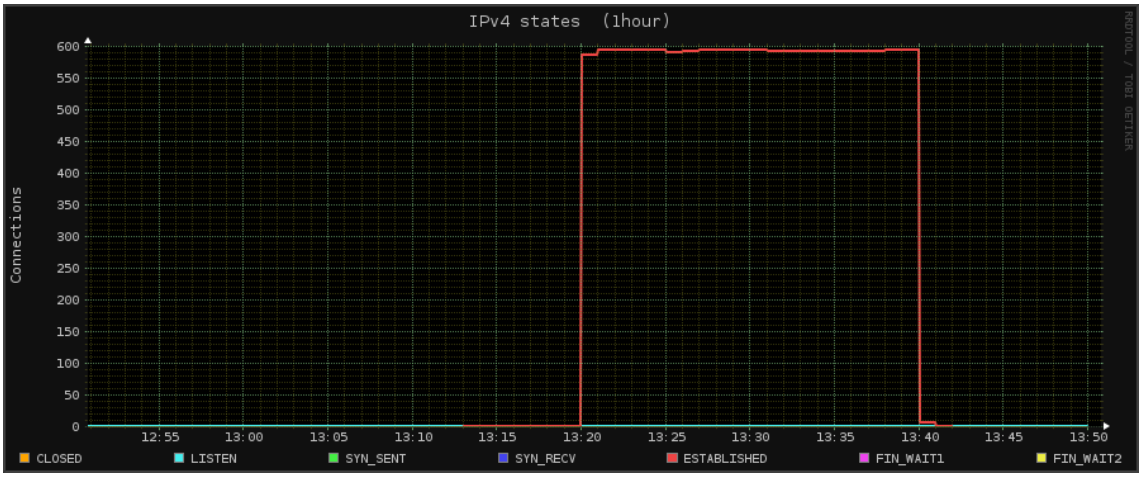
Obrázek 6.3: *Graf využití druhého jádra procesoru v nástroji Docker*

6.1.3 Využití operační paměti



Obrázek 6.4: *Graf využití operační paměti v nástroji Docker*

6.1.4 Využití sítě



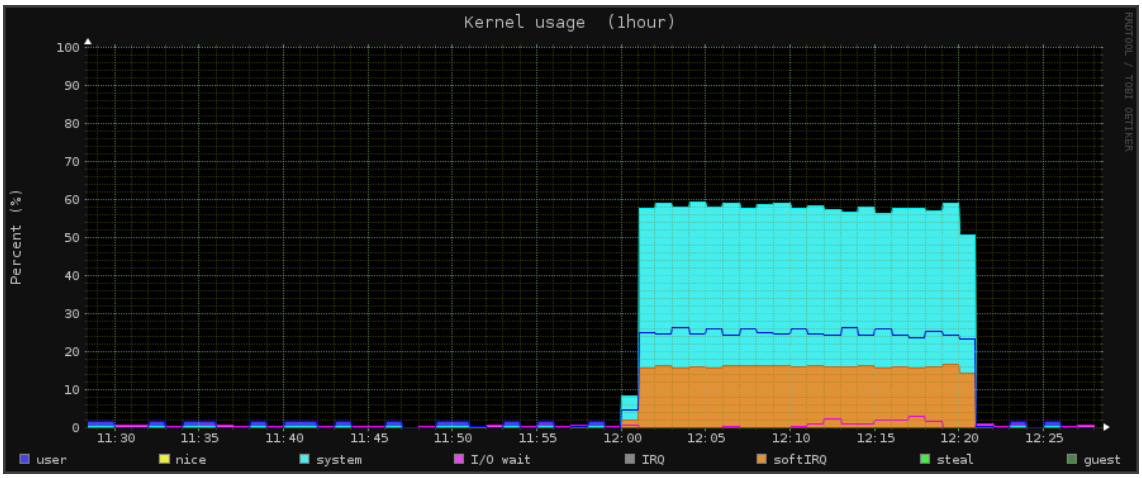
Obrázek 6.5: *Graf využití sítě v nástroji Docker*

6.2 LXC

Test byl spuštěn v úseku 12:00 - 12:20.

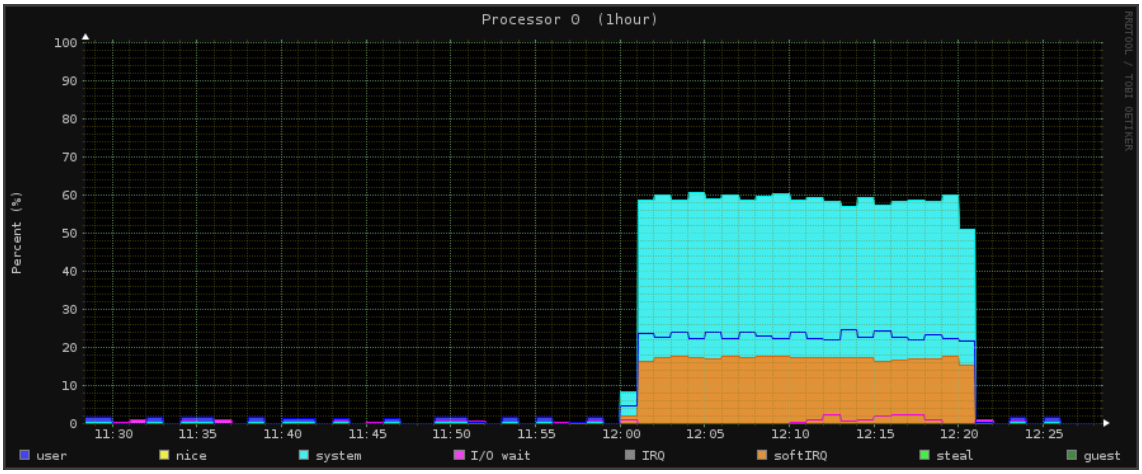
V následujících grafech lze vidět jednotlivé využití komponent v rámci daného nástroje během spuštěného výkonnostního testu. Konkrétně u grafu využití kernelu lze vidět, že nepřesahovalo hodnotu vyšší než je zhruba 59%, což je jen zanedbatelný pokles oproti Dockeru. U grafů využití jednotlivých jader procesoru ale můžeme vidět, že v tomto případě LXC mnohem lepé a rovnoměrně rozdělilo zátěž na obě jádra, než jak je tomu u Dockeru. Byly i o dost méně zatíženy a tak maximální hodnoty, kterých dosáhly, byly zhruba 60% a 58%. Využití operační paměti se dostalo zhruba na 1700MB. U sítě ale jak můžeme pozorovat, tak maximální počet spojení bylo pouze 380, a tuto hodnotu ani kontinuálně neudržoval vzhledem k znatelným výkyvům.

6.2.1 Využití kernelu

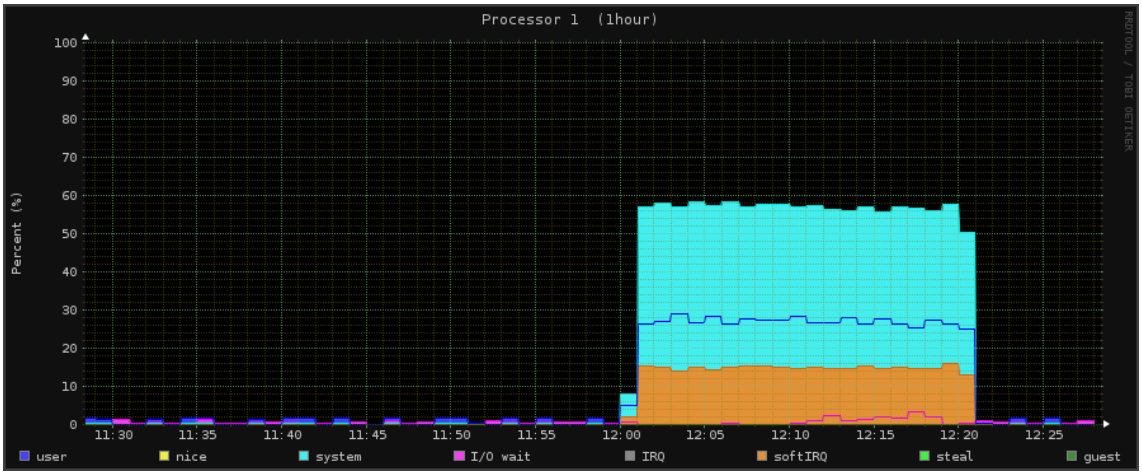


Obrázek 6.6: *Graf využití kernelu nástrojem LXC*

6.2.2 Využití jader procesoru

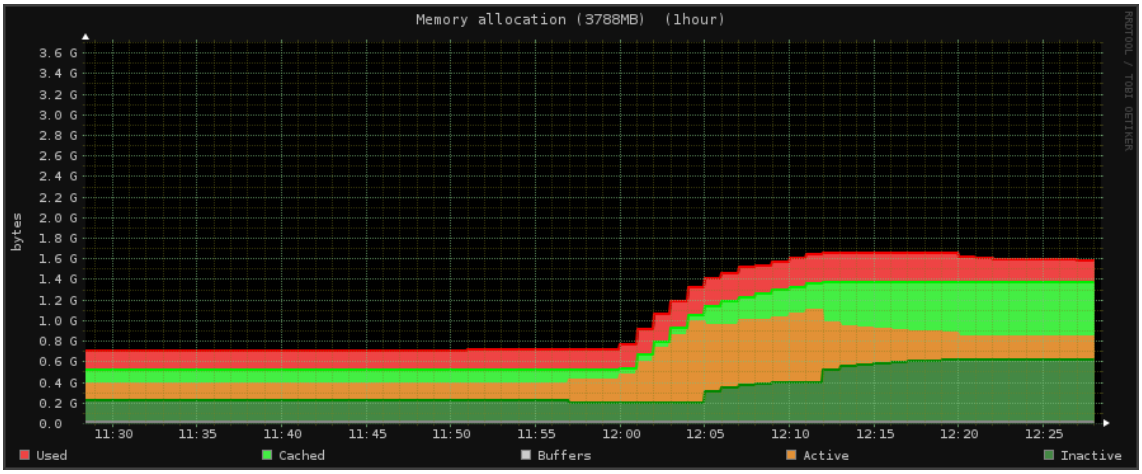


Obrázek 6.7: *Graf využití prvního jádra procesoru nástrojem LXC*



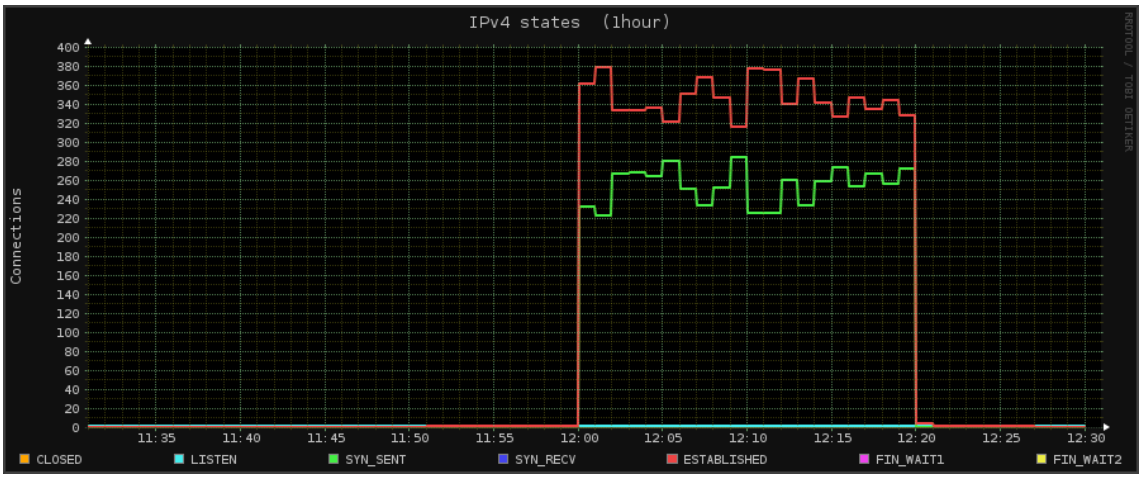
Obrázek 6.8: *Graf využití druhého jádra procesoru nástrojem LXC*

6.2.3 Využití operační paměti



Obrázek 6.9: *Graf využití operační paměti nástrojem LXC*

6.2.4 Využití sítě



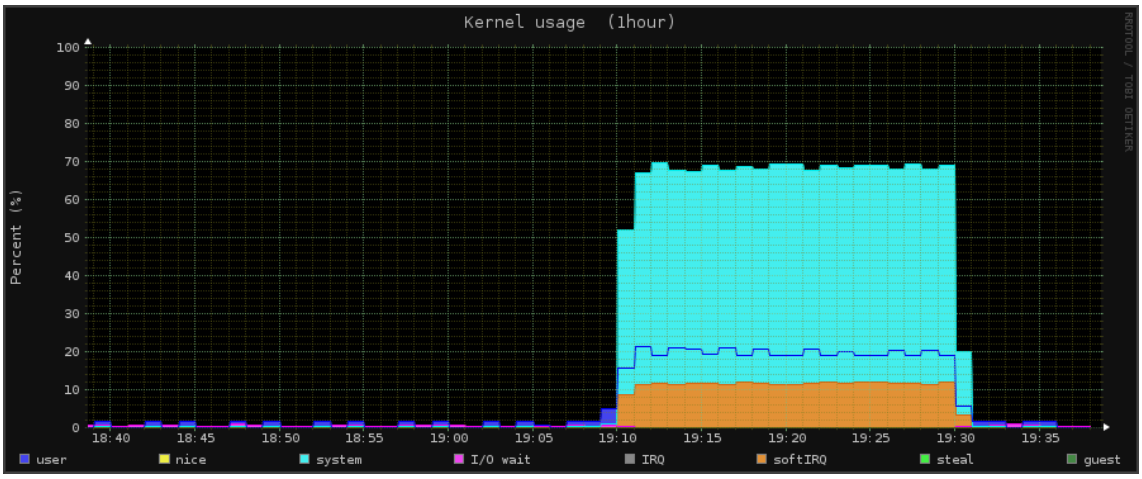
Obrázek 6.10: *Graf využití sítě nástrojem LXC*

6.3 LXD

Test byl spuštěn v úseku 19:10 - 19:30.

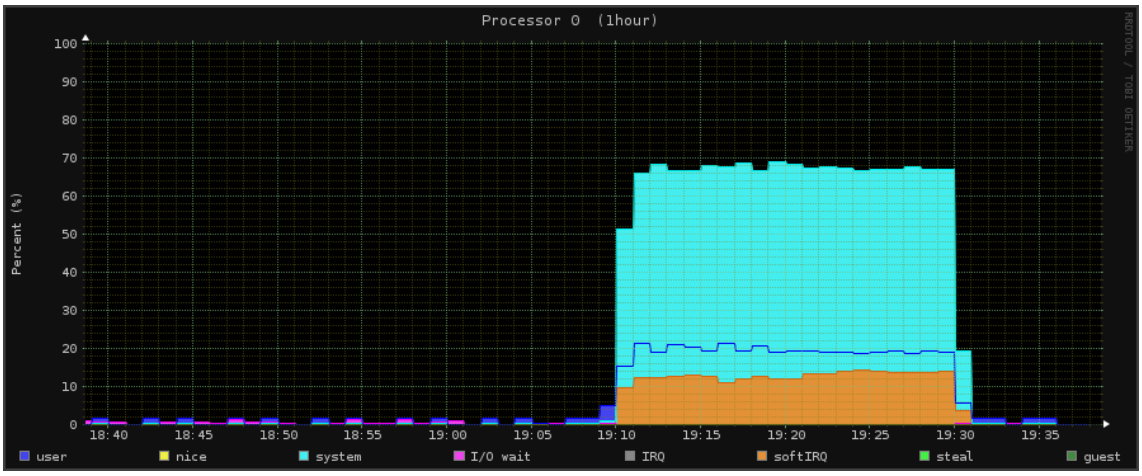
Na následujících výsledných grafech lze vidět jednotlivé využití komponent v rámci daného nástroje během spuštěného výkonnostního testu. U grafu využití kernelu lze vidět, že zde využití dosáhlo nejvyšší hodnoty oproti ostatním, a to konkrétně zhruba 69%. U grafů využití jednotlivých jader procesoru lze opět vidět, že LXD bylo také schopno mnohem lepé a rovnoměrně rozdělit zátěž na obě jádra, ale jejich využití dosahovalo zase o něco vyšších hodnot. Konkrétní maximální hodnoty, kterých dosáhly, byly 69% a 71%. V tomto případě i využití operační paměti se dostalo zhruba až na 2800MB, což je přikládáno hlavně tomu, že jak již bylo zmíněno, LXD je svým způsobem rozšíření pro LXC. U sítě lze opět pozorovat, že byly obrovské propady a maximální počet spojení bylo pouze 457.

6.3.1 Využití kernelu

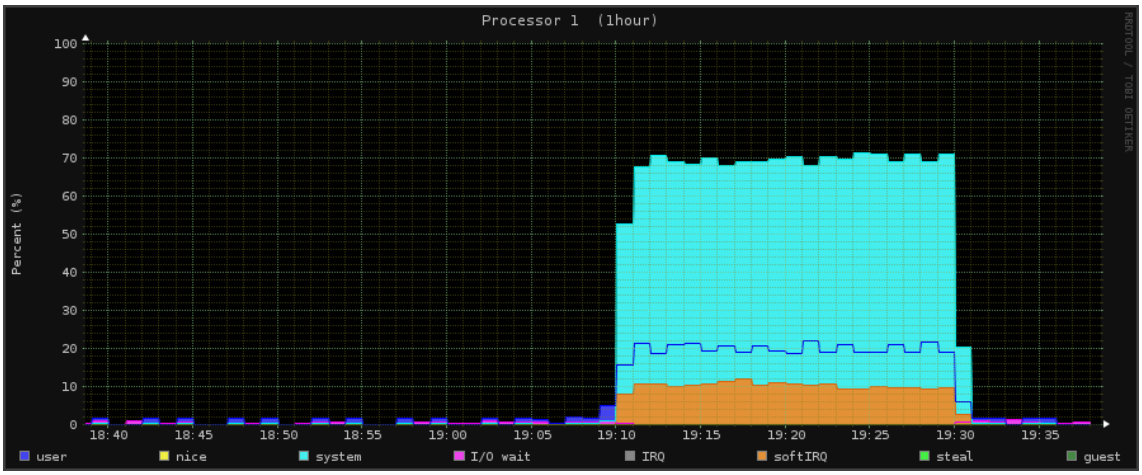


Obrázek 6.11: *Graf využití kernelu nástrojem LXD*

6.3.2 Využití jader procesoru

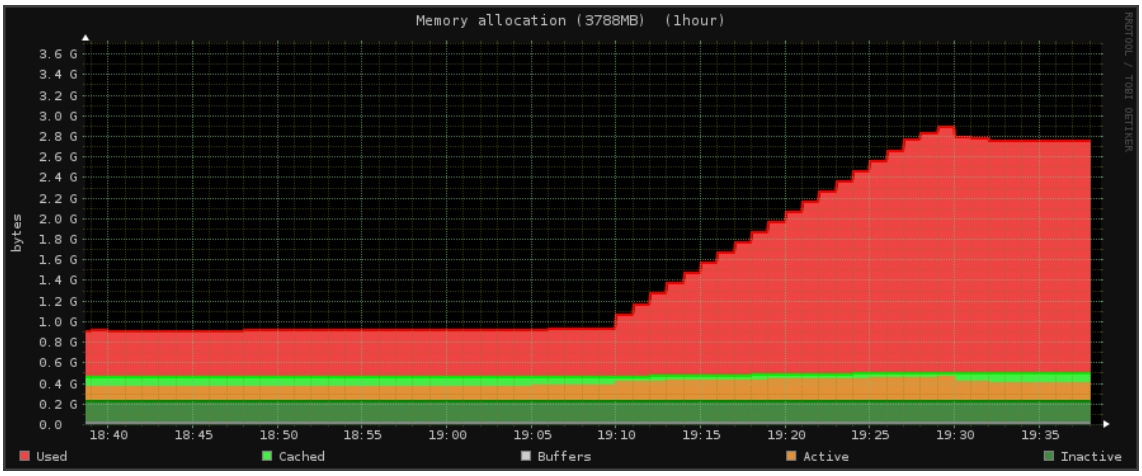


Obrázek 6.12: *Graf využití prvního jádra procesoru nástrojem LXD*



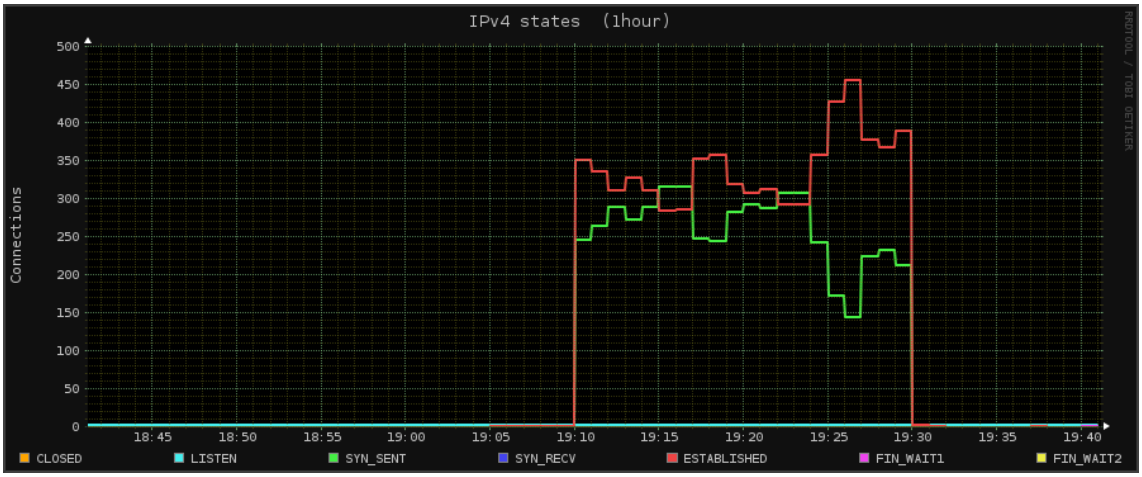
Obrázek 6.13: *Graf využití druhého jádra procesoru nástrojem LXD*

6.3.3 Využití operační paměti



Obrázek 6.14: *Graf využití operační paměti nástrojem LXD*

6.3.4 Využití sítě



Obrázek 6.16: *Graf využití sítě nástrojem LXD*

Závěr

Cílem této diplomové práce bylo navrhnout řešení postavené na kontejnerové virtualizaci v Linuxu a následně provést výkonnostní testování.

V teoretické části byly popsány různé virtualizační techniky a rozuzlení toho, kam vlastně kontejnerová virtualizace spadá. Dále byly v této části popsány jednotlivé užité nástroje pro kontejnerovou virtualizaci a jejich srovnání na základě zjištěných propagačních informací či programových dokumentací. Součástí je také rozbor několika monitorovacích nástrojů pro generování grafů jednotlivých fyzických zdrojů.

Praktická část byla vytvořena návrhem řešení a poté podrobným popisem instalace a konfigurace vybraných nástrojů pro kontejnerovou virtualizaci, společně s webovými servery a monitorovacím nástrojem. U každého nástroje bylo použito mírně odlišného postupu pro názornou ukázku různorodých řešení, avšak dodržení stanoveného návrhu. Pak už bylo pouze vše podrobeno výkonnostnímu testování.

Dohromady tato práce tvoří ucelený návod a popis na realizaci daného navrženého řešení pomocí nejpoužívanějších nástrojů pro kontejnerovou virtualizaci v Linuxu. Jelikož se tento směr virtualizace stále vyvíjí a je relativně nový, podobná celistvá práce zatím není ani na internetu k dispozici.

Zhodnocení výsledků výkonnostních testů vzhledem k řešení, lze brát dvěma způsoby. Buď jakožto posuzování pouze výkonu testovaných komponent či součástí, myšleno konkrétně jádra procesoru, operační paměť a kernel, byly tyto věci nejméně vytěžovány, neboli nejlepšího výkonu bylo dosaženo, nástrojem LXC. Naopak nejhoršího výkonu v tomto ohledu dosáhl nástroj LXD. Nebo posuzování vzhledem k použitým službám, tím jsou myšleny konkrétně webové servery. Jelikož tyto servery byly podrobeny benchmarku a šlo tak vlastně ve výsledku o to, který z nástrojů dokáže tyto služby nejlépe provozovat, za předpokladu nejlepšího užitého výkonu, jediným dá se říct úspěšným, se stal nástroj Docker. Jak lze vidět na výsledných grafech, jako jediný totiž dokázal požadovaný počet spojení udržet po celou dobu testu. Zatímco nástroje LXC, potažmo LXD, nejenže nebyly schopny daného počtu spojení dosáhnout, ale nebyly ani schopny si nějaký počet spojení kontinuálně udržet. Na grafech jsou viditelné výrazné výkyvy. Docker se tedy díky testům jeví jako nejlepší řešení pro provozování dvou webových serverů pomocí kontejnerů.

Z pohledu dalšího vývoje práce by bylo dobré otestovat také kombinaci různých služeb v rámci kontejnerů. Tím je myšleno například kombinace DHCP, DNS a FTP. Samozřejmě za použití mnohem výkonnějšího hardwaru a nejlépe vyzkoušení nasazení v reálném provozu oproti laboratorním podmínkám, kde by výsledky byly zase o to jiné. V tom případě by se naskytovala možnost vyzkoušení i mnou jiných avizovaných monitorovacích nástrojů.

Použitá literatura

[1] Emulation, paravirtualization and pass-through. Brianmadden [online]. [cit. 2018-04-18]. Dostupné z: https://www.brianmadden.com/opinion/Emulation-paravirtualization-and-pass-through-what-you-need-to-know-for-client-hypervisors

[2] Operating-system-level virtualization. Wikipedia [online]. [cit. 2018-04-18]. Dostupné z: https://en.wikipedia.org/wiki/Operating-system-level_virtualization

[3] Virtualizace v Linuxu. Wikibooks [online]. [cit. 2018-04-18]. Dostupné z: https://cs.wikibooks.org/wiki/Virtualizace_v_Linuxu

[4] Overview of Virtualization Technologies. Logicalread [online]. [cit. 2018-04-18]. Dostupné z: https://logicalread.com/virtualization-for-oracle-mc11/#.WtcZVH9pHIV

[5] Docker Documentation [online]. [cit. 2018-04-18]. Dostupné z: https://docs.docker.com/

[6] LXC Introduction. Linux Containers [online]. [cit. 2018-04-18]. Dostupné z: https://linuxcontainers.org/lxc/introduction/

[7] LXC. Wikipedia [online]. [cit. 2018-04-18]. Dostupné z: https://en.wikipedia.org/wiki/LXC

[8] LXD Introduction. Linux Containers [online]. [cit. 2018-04-18]. Dostupné z: https://linuxcontainers.org/lxd/

[9] The LXD container hypervisor. Ubuntu [online]. [cit. 2018-04-18]. Dostupné z: https://www.ubuntu.com/containers/lxd

[10] Grafana features. Grafana [online]. [cit. 2018-04-18]. Dostupné z: https://grafana.com/grafana#visualize

[11] Kibana. Wikipedia [online]. [cit. 2018-04-18]. Dostupné z: https://en.wikipedia.org/wiki/Kibana

[12] Monitorix a lightweight system and network monitoring tool for linux. Tecmint [online]. [cit. 2018-04-18]. Dostupné z: https://www.tecmint.com/monitorix-a-lightweight-system-and-network-monitoring-tool-for-linux/

[13] LXC (LinuX Containers) Introduction. Support.Tux4u [online]. [cit. 2018-04-18]. Dostupné z: http://support.tux4u.nl/dokuwiki/doku.php?id=linux:applicaties:lxc:lxc-introduction

[14] Use Macvlan networks. Docs.Docker [online]. [cit. 2018-04-18]. Dostupné z: https://docs.docker.com/network/macvlan/

[15] Exploring LXC Networking. Containerops [online]. [cit. 2018-04-18]. Dostupné z: http://containerops.org/2013/11/19/lxc-networking/

[16] Install Docker. Docs.Docker [online]. [cit. 2018-04-18]. Dostupné z: https://docs.docker.com/install/

[17] Limit a container's resources. Docs.Docker [online]. [cit. 2018-04-18]. Dostupné z: https://docs.docker.com/config/containers/resource_constraints/

[18] Use NGINX As A Reverse Proxy To Your Containerized Docker Applications. Thepolyglotdeveloper [online]. [cit. 2018-04-18]. Dostupné z: https://www.thepolyglotdeveloper.com/2017/03/nginx-reverse-proxy-containerized-docker-applications/

[19] LXC Getting started. Linuxcontainers [online]. [cit. 2018-04-18]. Dostupné z: https://linuxcontainers.org/lxc/getting-started/

[20] Setting CPU Resource Limits With LXC. Jamescoyle [online]. [cit. 2018-04-18]. Dostupné z: https://www.jamescoyle.net/how-to/2532-setting-cpu-resource-limits-with-lxc

[21] Setup Linux Container with LXC on Ubuntu 16.04. Itzgeek [online]. [cit. 2018-04-18]. Dostupné z: https://www.itzgeek.com/how-tos/linux/ubuntu-how-tos/setup-linux-container-with-lxc-on-ubuntu-16-04-14-04.html

[22] LXD Installation. Linuxcontainers [online]. [cit. 2018-04-18]. Dostupné z: https://linuxcontainers.org/lxd/getting-started-cli/

[23] LXD 2.0: Resource control [4/12]. Stéphane Graber's website [online]. [cit. 2018-04-18]. Dostupné z: https://stgraber.org/2016/03/26/lxd-2-0-resource-control-412/

[24] How to Host Multiple Web Sites with Nginx and HAProxy Using LXD on Ubuntu 16.04. DigitalOcean [online]. [cit. 2018-04-18]. Dostupné z: https://www.digitalocean.com/community/tutorials/how-to-host-multiple-web-sites-with-nginx-and-haproxy-using-lxd-on-ubuntu-16-04#prerequisites

[25] Installation on a Debian/Ubuntu Linux. Monitorix [online]. [cit. 2018-04-18]. Dostupné z: http://www.monitorix.org/doc-debian.html

[26] Wrk - a HTTP benchmarking tool. Github [online]. [cit. 2018-04-18]. Dostupné z: https://github.com/wg/wrk

Seznam příloh

Součástí BP/DP je CD/DVD.

Adresářová struktura přiloženého CD/DVD: